Conjunctive and disjunctive version spaces with instance-based boundary sets

Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van de Rector Magnificus,
Prof. dr. A.C. Nieuwenhuijzen Kruseman,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op donderdag 22 februari 2001, om 16.00 uur

door

Evgueni Nikolaevich Smirnov

Promotor: Prof. dr. H.J. van den Herik

Leden van de beoordelingscommissie:
Prof. dr. A.J. van Zanten (voorzitter)
Prof. ir. L.A.A.M. Coolen
Prof. dr. T.M. Mitchell (Carnegie Mellon University, Pittsburgh)
Dr. ir. J. Talmon
Prof. dr. B.J. Wielinga (Universiteit van Amsterdam)

SIKS Dissertation Series No. 2001-4
ISBN 90-423-0146-5
Shaker Publishing

Printed in The Netherlands

Shaker Publishing BV
St. Maartenslaan 26
6221 AX Maastricht
Tel.: 043-3500424
Fax: 043-3255090
http://www.shaker.nl/
Cover design: Shaker Publishing

Contents

Preface

I am grateful to the Institute for Knowledge and Agent Technologies (IKAT) for the opportunity to continue my research that resulted in this thesis. It is IKAT's first Ph.D. thesis in the third millennium.

I would like to thank my supervisor, Jaap van den Herik, for his continuous support, encouragement, and confidence. He guided my research in the last two years, and in that he showed me how to do scientific research and writing.

Second, I thank Ida Sprinkhuizen-Kuyper. She has strongly influenced my research through many fruitful discussions and intense cooperation. I am grateful for her restless willingness to read this work several times, each time coming up with more ideas and improvements. She always insisted on clarity, correctness, and structure, and made them cornerstones of this thesis.

I thank Eric Postma, Jacques Lenting, and Levente Kocsis for being with me in difficult times of the research and thesis writing. Eric's catching optimism and the ability "to avoid obstacles" made life and work much easier. Jacques' desire to help brought me all "version-space" experts in Maastricht that "forced the instance-based boundary sets to show their ability to collapse" in a very short period. Levente's need to speak about classical music and literature "saved" the research line presented in chapter 8.

I thank Floris Wiesman and Jeroen Donkers for sharing their experience in Java and LaTeX. This facilitated the experimental study presented in the thesis as well as the preparation of the final manuscript.

Many people provided relevant literature, literature pointers, or valuable advise. I thank, next to those already mentioned, Peter Andras, Natascha Bourdonskaia, Denis Breuker, Leo Coolen, Sandro Etalle, Peter Geurtz, Rens Kortmann, Paul van der Krogt, Leonard Plugge, Robert Plompen, Ruud van der Pol, Yongping Ran, Nico Roos, Arno Sprinkhuizen, Jos Uiterwijk, Joop Verbeek, Erik van der Werf, Tony Werten, Menthy Willems, and Jan van Zanten for their attentiveness.

I would like to acknowledge the organisational heart of our Institute: Joke Hellemons, Marlies van der Mee, Anjes Schreurs, Martine Tiessen, and Sabine Vanhouwe. Over the years they took many weights off my shoulders, and gave me a pleasant time. I thank Joke for her organisation of the Ph.D. procedure and defence. I am

grateful to Sabine for her help and advise when I settled in Maastricht.

Beyond IKAT many other people contributed to the thesis. My research with Nikolay Nikolaev revealed many drawbacks of the disjunctive version spaces; they were used when formulating the problem statement of this thesis. My cooperation with Peter Braspenning actually started the research on the instance-based boundary sets. Michele Sebag taught me first lessons on how to make version spaces practical. The correspondence with Haym Hirsh improved my understanding of the version-space representations (including my own). Dries Vermeulen helped me to specify the classes of concept languages applicable to the instance-based boundary sets. Discussions with Tom Mitchell helped me to evaluate my research and to continue with more practical version-space approaches. Sergey Kuznetsov showed me how to investigate the properties of the admissible concept languages.

I also want to thank my parents for their confidence and support. Finally, I wish to thank my wife Polly for her love: she looked after me when I was writing the thesis during all those long evenings.

Acknowledgements
The research reported in this thesis is supported, in part, by the Netherlands Organisation for Scientific Research (NWO). It has been carried out under auspices of SIKS, the Dutch Research School for Information and Knowledge Systems. In the final stage the support by KPN Research is gratefully acknowledged.

Evgueni Smirnov Maastricht, January 2001

Chapter 1

Introduction

This thesis is in the field of machine learning. It presents research on version spaces in the context of the concept-learning task. In this chapter the task is briefly sketched and version spaces are informally introduced. Four principal problems of version spaces are emphasised and formulated in the thesis's problem statement, which determines the research lines described in the subsequent chapters.

1.1 Concept-Learning Task

The concept-learning task occupies a central position in the field of machine learning (Michalski, Carbonell, and Mitchell, 1983, 1986; Michalski and Kodratoff, 1990; Shavlik and Dietterich, 1990; Michalski and Tecuci, 1994; Langley, 1996; Mitchell, 1997). It is defined under the assumption that concepts are sets of entities unified by a reason (Mechelen *et al.*, 1993). The elements of concepts are called instances. They are positive for a concept if they belong to the concept; otherwise they are negative. Instances are represented by their descriptions in an instance language. The descriptions of all the instances of a concept form the extensional representation of the concept. Since extensional representations can be large or even infinite, concepts are studied in a concept language. The concept language is a set of descriptions that represent concepts intensionally. A description is intensional for a concept if there exists a rule that maps the description to the concept. The concept language is related to the instance language by a membership relation. The relation holds for a concept description and an instance description if and only if the instance description stands for an instance that is member of a concept given by the concept description.

Given an instance language, a concept language, and a membership relation, the concept-learning task is to find intensional descriptions of some unknown target concept from sets of its positive and negative training instances expressed in the instance language.

1.2 Version Spaces

The descriptions of any target concept have to be specified by acceptance criteria. One of the simplest criteria is the consistency criterion (Mitchell, 1978). It states that a description is consistent with the training instances of a target concept if the description correctly classifies those instances. When this criterion is employed the complete solution of the concept-learning task is a version space (Mitchell, 1978, 1982, 1997; Hirsh, 1989). Hence, this version space is defined as a set of all descriptions in the concept language that are consistent with the training instances of the target concept.

Version-space learning can be viewed as a complete search in the concept language according to constraints imposed by the training instances (Mitchell, 1978). The search is specified by three operations. The first operation, *Initialise*, sets version spaces when no training instances are available. If a new instance is added to the sets of the training instances, the second operation, *Update*, removes descriptions from version spaces that incorrectly classify the instance (Mitchell, 1978; Hirsh, 1992b). Conversely, if an instance is removed from the sets of the training instances, the third operation, *Retraction*, adds those descriptions to version spaces that incorrectly classify the instance and are consistent with the remaining training instances (Idemstam-Almquist, 1990; Hirsh, Mishra, and Pitt, 1997). The status of the search can be identified by two state-test operations *Collapsed?* and *Converged?* (Mitchell, 1978; Hirsh, 1992b). The operations determine when version spaces are empty and when they consist of equivalent concept descriptions. The instance classification is realised by the operation *Classify*. This operation is based on the rule of the unanimous vote of the descriptions in version spaces (Mitchell, 1978; Hirsh, 1992b). An instance is classified as positive if all the descriptions in a version space do cover the instance; it is classified as negative if all the descriptions in the version space do not cover the instance; otherwise, the instance cannot be classified. Since version spaces are essentially sets, they can be additionally characterised by set operations *Member?*, *Intersection*, *Subset?*, and *Equal?* (Hirsh, 1992b).

The definition of version spaces together with their basic operations (described above) can be considered as an abstract data type. Given a concept language an implementation of the version-space abstract data type requires designing (1) a version-space representation, and (2) algorithms of the basic version-space operations based on that representation (Hirsh, 1992b). A version-space representation is a finite data structure that contains "the information needed to reconstruct every concept description in the version space" (Mitchell, 1978, p.25). Given a concept language, a version-space representation can have two types of adequacy for the basic version-space operations (Hirsh, 1992b). The representation is epistemologically adequate if each operation can be implemented by an algorithm based on the representation. The representation is heuristically adequate if the algorithm of each operation is tractable. An algorithm is tractable if its time complexity is polynomial in the size of the input and relevant properties of the concept language. The heuristical ad-

equacy of a representation (for the basic version-space operations) is closely related to the tractability of the representation. A version-space representation is tractable for a concept language if the representation can be computed in time polynomial in the number of the training instances and relevant properties of the language.

The standard version-space representation is by boundary sets (Mitchell, 1978). The representation presupposes that the relative generality of descriptions in concept languages imposes a partial order. Therefore, the boundary sets are defined to contain the minimal and maximal descriptions in version spaces. This implies that they delimit version spaces in concept languages. The epistemological adequacy of the boundary-set representation with respect to the basic version-space operations was established for the class of admissible concept languages (Mitchell, 1978; Hirsh, 1992b). Nevertheless, Haussler (1988) showed that the representation is intractable for most concept languages.

The intractability of the boundary sets is one of the four principal problems of version spaces. We systematically consider these problems in the rest of this section.

Problem with Incomplete Concept Languages. A concept language is incomplete if at least one concept, extensionally defined in the instance language, is not represented in the language. Mitchell (1997) showed that if a concept language is incomplete and the description of a target concept is not in the language, then the version space is empty. One way to overcome this problem is to extend the concept language with logical disjunction (Mitchell, 1978). It was shown that the disjunctive extension of a concept language can represent more concepts than the language itself. This observation was used in the extended version-space approach (Mitchell, 1978). The approach maintains in parallel a set of version spaces consistent with different subsets of the training instances determined by preliminary given parameters. Depending on the parameters the set can represent target concepts in disjunctive extensions of concept languages[1]. Since the concepts are not known in advance, the approach is sensitive with respect to the parameters. Moreover, the approach is not tractable for most concept languages: instead of updating one version space, a set of version spaces, represented by boundary sets, is updated. Thus, the approach is not practical, it has only theoretical value[2].

To overcome the problem of the parametric dependence of the extended version-space approach Murray (1987) developed a disjunctive version-space approach. The approach was proposed for learning concepts in disjunctive extensions of concept languages. Disjunctive version spaces were defined as sets of maximally complete version spaces combined by logical disjunction. A version space is maximally complete if it is a set of descriptions in the concept language used that are consistent with a maximal number of the positive training instances and all the negative train-

[1] By duality we have found that the extended version-space approach can be used for learning concepts in the conjunctive extensions of concept languages.

[2] In spite of the critiques the extended version-space approach was used in the Meta-DENDRAL program (Mitchell and Schwenger, 1978).

ing instances. The approach was generalised by Sablon (1995), Smirnov and Neves (1998) for the class of admissible concept languages. Its worst-case complexity analysis showed that it is still intractable for most concept languages. In spite of that Smirnov and Neves (1998) reported that the complexity of the approach is acceptable for some concept-learning tasks from the Machine Learning Database Repository at the University of California, Irvine (Blake and Merz, 1998).

Computational Problem. As mentioned above the standard version-space representation by boundary sets is intractable for most concept languages (Haussler, 1988). This holds even for simple 1-CNF or 1-DNF languages with Boolean attributes (Hirsh *et al.*, 1997). To overcome this computational problem alternative version-space representations were introduced by Idemstam-Almquist (1990), Smith and Rosenbloom (1990), Hirsh (1992b), Sablon (1995), Sebag (1996), Sebag and Rouveirol (1997, 2000). The alternative representations extend the computational applicability of version spaces even to concept languages for which version spaces originally were not considered. Hirsh *et al.* (1997) analysed the alternative representations and they found that version-space learning is partially equivalent to the consistency problem. The consistency problem is to determine the existence of a concept description that correctly classifies training instances. Therefore, the training instances themselves were proposed as a version-space representation. The representation is tractable and that is why it stimulates the research on tractable algorithms for the consistency problem.

Retraction Problem. The principal difficulty of the alternative version-space representations is that they were not designed for the operation *Retraction*. That is why, in general, they are computationally inefficient for this operation. The only exception is the work of Idemstam-Almquist (1990). He proposed a version-space representation for conjunctive languages with tree-structured attributes that is heuristically adequate for the operation *Retraction*. Unfortunately, the representation is language-dependent and that is why it was not generalised for broader classes of concept languages.

Problem with Noisy Training Instances. A training instance is noisy if it is incorrectly classified with respect to the target concept. Mitchell (1997) showed that, if at least one of the training instances is noisy, the description of the target concept is removed from the version space to be learned. To overcome this problem Mitchell (1978) suggested to use his extended version-space approach that, as mentioned before, is not practical. Therefore, Hirsh (1989) proposed a version-space approach for learning from instances with bounded inconsistency. "The underlying assumption for this class of inconsistency is that some small perturbation to the description of any bad instance will result in a good instance" (Hirsh, 1989, p.25). That is why when a new training instance is given, it is perturbed in such a way that all its possible interpretations in the context of the current concept-learning task are found. The approach updates version spaces by removing descriptions that are

inconsistent with all the instance interpretations. If in some point of the concept-learning process the version spaces are not empty, then they contain descriptions that are consistent with at least one chain of interpretations of all the training instances. Therefore, Hirsh's approach can handle training instances with bounded inconsistency. A complexity analysis shows that the approach can be practical since it overcomes some of the computational problems of the extended version-space approach, but not all. Moreover, in order to use the approach we have to know how to perturb instances. This is not a trivial task that varies from domain to domain.

Sebag (1996) addressed the version-space problems analysed above with the exception of the retraction problem. She introduced a new type of disjunctive version spaces. They were defined as sets of version spaces for positive instances combined by logical disjunction. A version space for a positive training instance is a set of all concept descriptions consistent with that instance and all the negative training instances. Version spaces for positive instances were represented with a special layered representation that is tractable for conjunctive attributive languages. The instance classification was proposed for an arbitrary number of target concepts. An instance is classified to belong to a concept if among all version spaces which layered representations cover the instance, the version spaces for positive instances of that concept are the majority. The majority vote can be additionally adjusted by two preliminarily-tuned parameters that correspond to the level of disjunctiveness of the target concepts as well as the level of noise in the training instances. Sebag showed that when the parameters are properly chosen her disjunctive version spaces have a good classification performance for concept-learning tasks with noisy training instances.

1.3 Problem Statement

From the previous section we have found that:

- Version spaces can be applied in the presence of incomplete concept languages if the targets concepts are presented in disjunctive extensions of those languages. If not, we can apply other logical operators, a direction of research that has not been carried out in the context of version spaces.

- The alternative version-space representations overcome the computational problem of the boundary sets even for concept languages for which version spaces originally have not been considered. Nevertheless, they are computationally inefficient for the operation *Retraction*. An exception is the work of Idemstam-Almquist (1990) which is, however, applicable only for a restricted class of concept languages.

- Version spaces in their pure form cannot handle noisy instances because of the requirement for strong consistency with training data.

Given the findings above we formulate our problem statement:

Do there exist version spaces and their corresponding representations that can simultaneously solve:

(1) the problem with incomplete concept languages;

(2) the computational problem;

(3) the retraction problem; and

(4) the problem with noisy training instances?

This thesis is an attempt to find answers to the problem statement. The next chapters describe them in detail.

1.4 Thesis Outline

The thesis is organised as follows. Chapter 2 formalises the concept-learning task. The task is viewed as a search task. In this context the notion of inductive bias is introduced.

Chapter 3 proposes to consider version spaces as an abstract data type. The data type is specified by the definition of version spaces and their set of basic operations.

Chapter 4 tries to answer the first part of the problem statement. It introduces conjunctive and disjunctive extensions of concept languages. Their completeness properties are analysed. The analysis is used as a premise for introducing conjunctive and disjunctive version spaces. Their completeness properties are analysed as well. It is shown that conjunctive and disjunctive version spaces are a partial solution to the problem with incomplete concept languages. Following chapter 3 conjunctive and disjunctive version spaces are considered as abstract data types. It is proven that the conjunctive and disjunctive version-space abstract data types can be implemented using the version-space abstract data type.

Chapter 5 considers the problem of implementing the version-space abstract data type. In this context it surveys three existing version-space representations: (1) list representation, (2) boundary sets, and (3) unilateral boundary sets. The representations are presented in terms of their epistemological and heuristical adequacy for the basic version-space operations as well as in terms of their tractability. They are improved by developing their algorithms of those basic version space operations which implementation details were not considered previously.

Chapter 6 answers the second and third part of the problem statement. It proposes a new alternative version-space representation, called instance-based boundary sets. The epistemological adequacy of the representation is proven with respect to a subset of the basic version-space operations including the operation *Retraction* for the classes of intersection-preserving and union-preserving languages, introduced in

the chapter. The conditions for the tractability and the heuristical adequacy of the representation for the same subset of operations are derived. An analysis of these conditions shows that they can easily be met in practice. Therefore, it is shown that the instance-based boundary sets can be used for efficient implementations of the version-space abstract data type. Moreover, it is concluded that they can provide a solution to the computational and retraction problems of version spaces.

Chapter 7 answers the first, second, and third part of the problem statement. It shows that the representations from chapters 5 and 6 can represent conjunctive and disjunctive version spaces and are epistemologically adequate with respect to the basic conjunctive and disjunctive version-space operations (for the concept languages for which they have been designed). Therefore, it is concluded that the conjunctive and disjunctive version-space abstract data types can be implemented using each of these representations. It is determined when the representations are heuristically adequate for the basic conjunctive and disjunctive version-space operations, and when they tractably represent conjunctive and disjunctive version spaces. In this context conjunctive and disjunctive version spaces represented by instance-based boundary sets are considered as a solution to the problem with incomplete concept languages, the computational problem, and the retraction problem of version spaces.

Chapter 8 answers the fourth part of the problem statement. It proposes to apply an adaptation of the k-nearest-neighbour algorithm on the instance-based boundary sets of conjunctive and disjunctive version spaces. This results in two instance-based classification schemes. The schemes do not require any change in the instance-based boundary-set representation of conjunctive and disjunctive version spaces. Hence, they can be added to the conjunctive and disjunctive version-space abstract data types. The systems, based on the classification schemes, are tested on standard datasets. Their classification performance is comparable with those of the C4.5 decision-tree learner. The systems are combined with the condensed nearest-neighbour algorithm and the edited nearest-neighbour algorithm. The first combination aims at reduction of the computational complexity, but the experimental results show a significant drop in the classification performance. The second combination aims to achieve robustness of the classification performance with respect to noise in training instances. Such robustness is confirmed by experimental results. Hence, the problem with noisy training instances is solved.

Chapter 9 provides an evaluation of the problem statement, final conclusions, and future research directions.

Appendices A and B consist of theorems for correctness of the algorithms of the basic version-space operations based on the boundary sets and the unilateral boundary sets, respectively. Appendix C contains a concept-learning task used in the experiments with the algorithms of the basic version-space operations based on the instance-based boundary sets.

Chapter 2

Concept-Learning Task

This chapter formalises the concept-learning task considered in the thesis. In section 2.1 we introduce the elements of the task: instance language, concept language, membership relation, and sets of training instances of a target concept. The goal of the task is specified in section 2.2: it is to find descriptions of the target concept in the concept language that accurately classify instances represented in the instance language. To find such descriptions acceptance criteria are required. We propose to employ the consistency criterion that is used in a complete formalisation of the concept-learning task in subsection 2.2.1. The task itself is considered in different dimensions in subsection 2.2.2. We distinguish online and offline tasks, tasks in the presence of concept drift, and tasks characterised by noisy training instances.

In section 2.3 we consider a way to solve the concept-learning task. We employ the idea that concept learning can be viewed as a search in concept languages, that is based on constraints imposed by the training instances. In this context we introduce the notion of inductive bias in section 2.4.

2.1 Elements

To formalise the concept-learning task we determine its elements according to (Mitchell, 1978; Sablon, 1995). We start with the set of possible entities that we can encounter in a domain of discourse.

Notation 2.1. *The set of all entities is denoted by* $\mathfrak{I}$.

The concept-learning task has sense when the set $\mathfrak{I}$ is not empty. Hence, the set is restricted in constraint 2.2.

Constraint 2.2. *The set* $\mathfrak{I}$ *of all instances is nonempty.*

Having the set $\mathfrak{I}$ we extensionally define the notion of concepts.

Definition 2.3. (Concept) *A concept is a set of entities in* $\mathfrak{I}$.

Notation 2.4. *The set of all possible concepts in* $\mathfrak{I}$ *is denoted by* $\mathfrak{C}$.

The elements of the concepts $\mathfrak{c} \in \mathfrak{C}$ are called instances. To refer to and to reason with instances we represent them in a language.

Definition 2.5. (Instance Language) *The instance language Li is a set of descriptions.*

To associate an instance $\mathfrak{i} \in \mathfrak{I}$ with a description $i \in Li$ we define a bijective function $\mathcal{R}_{\mathfrak{I}}$.

Definition 2.6. *The bijective function* $\mathcal{R}_{\mathfrak{I}} : Li \rightarrow \mathfrak{I}$ *maps an instance description* $i \in Li$ *to the instance* $\mathfrak{i} \in \mathfrak{I}$ *it represents.*

Since $\mathcal{R}_{\mathfrak{I}}$ is a function no two distinct instances in $\mathfrak{I}$ can be represented by the same description in Li. The fact that $\mathcal{R}_{\mathfrak{I}}$ is bijective implies that $\mathcal{R}_{\mathfrak{I}}$ is surjective and injective[1]. Hence,

(1) every instance $\mathfrak{i} \in \mathfrak{I}$ is represented in Li since $\mathcal{R}_{\mathfrak{I}}$ is surjective; and

(2) every instance $\mathfrak{i} \in \mathfrak{I}$ is represented with exactly one description $i \in Li$ since $\mathcal{R}_{\mathfrak{I}}$ is injective.

From (1) and (2) it follows that every concept can be uniquely represented with a set of descriptions in the instance language Li if the descriptions represent all concept's instances. This concept representation is called extensional and it is formalised below.

Definition 2.7. (Extensional Concept Representation) *The extensional representation of a concept* $\mathfrak{c} \in \mathfrak{C}$ *is defined as follows:*

$$\{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}\}.$$

The extensional representations are unique for concepts. This property is shown in corollary 2.8. The corollary states that the extensional representations of each two concepts are equal if and only if the concepts are equal.

Corollary 2.8. *For all* $\mathfrak{c}_1, \mathfrak{c}_2 \in \mathfrak{C}$*:*

$$\{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}_1\} = \{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}_2\} \leftrightarrow \mathfrak{c}_1 = \mathfrak{c}_2.$$

[1] Note that constraint 2.2 and the bijectiveness of $\mathcal{R}_{\mathfrak{I}}$ imply that Li is nonempty.

The extensional concept representations are not compact (Langley, 1996). Moreover, they can be infinite. Hence, we need to represent concepts intensionally. Informally, a concept is intensionally represented if it is defined by a description (name) and there is a rule that maps the description to the concept's instances.

To represent concepts intensionally we introduce a language of concepts.

Definition 2.9. (Concept Language) *The concept language Lc is a set of descriptions c.*

To associate a concept $\mathfrak{c} \in \mathfrak{C}$ with a description $c \in Lc$ we define an injective function $\mathcal{R}_{\mathfrak{C}}$.

Definition 2.10. *The injective function $\mathcal{R}_{\mathfrak{C}} : Lc \rightarrow \mathfrak{C}$ maps a concept description $c \in Lc$ to the concept $\mathfrak{c} \in \mathfrak{C}$ it represents.*

Since $\mathcal{R}_{\mathfrak{C}}$ is a function, no two distinct concepts in $\mathfrak{C}$ can be represented by the same description in Lc. Since $\mathcal{R}_{\mathfrak{C}}$ is injective, no two distinct descriptions in Lc can represent the same concept in $\mathfrak{C}$.

We do not require the function $\mathcal{R}_{\mathfrak{C}}$ to be surjective. This has two consequences. The first one is that Lc can be empty. Hence, we restrict Lc in constraint 2.11.

Constraint 2.11. *The concept language Lc is nonempty.*

The second consequence is that it is possible to have concepts that are not represented in the concept language Lc. Therefore, the function $\mathcal{R}_{\mathfrak{C}}$ is closely related to the notion of (in)completeness of concept languages. A concept language Lc is complete with respect to the set $\mathfrak{C}$ of all possible concepts if and only if every concept $\mathfrak{c} \in \mathfrak{C}$ is represented in Lc. This is formalised in definition 2.12 below.

Definition 2.12. (Completeness of Concept Languages) *A concept language Lc is complete with respect to the set $\mathfrak{C}$ of all possible concepts if and only if:*

$$(\forall \mathfrak{c} \in \mathfrak{C})(\exists c \in Lc)(\mathcal{R}_{\mathfrak{C}}(c) = \mathfrak{c}).$$

A concept language Lc is complete if and only if the function $\mathcal{R}_{\mathfrak{C}}$ is not only injective but it is also surjective; i.e., it is bijective. This is given as corollary 2.13.

Corollary 2.13. *A concept language Lc is complete with respect to the set $\mathfrak{C}$ of all possible concepts if and only if $\mathcal{R}_{\mathfrak{C}}$ is bijective.*

When concept languages are incomplete we still want to compare them in terms of completeness. We say that a concept language is more complete than another if it represents more concepts from the set $\mathfrak{C}$ of all possible concepts.

Instances are related to concepts by the membership relation. The relation has to be projected into a membership relation between instance and concept descriptions. Following (Mitchell, 1978) the latter is denoted by M.

Definition 2.14. (The membership relation M)
$M : Lc \times Li \rightarrow Boolean$
defined by: $M(c,i) \leftrightarrow \mathcal{R}_{\mathfrak{I}}(i) \in \mathcal{R}_{\mathfrak{C}}(c)$.

Definition 2.14 states that the relation $M(c,i)$ holds for $c \in Lc$ and $i \in Li$ if and only if the instance $\mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{I}$ is a member of the concept $\mathcal{R}_{\mathfrak{C}}(c) \in \mathfrak{C}$. We say that a description $c \in Lc$ covers a description $i \in Li$ if and only if $M(c,i)$ holds. Conversely, we say that a description $c \in Lc$ does not cover (rejects) a description $i \in Li$ if and only if $M(c,i)$ does not hold.

Using this terminology we redefine the notion of extensional representation of a concept when the concept is given with a description $c \in Lc$ (see corollary 2.15). In this case the extensional representation of the concept is the set of descriptions $i \in Li$ that are covered by the description c.

Corollary 2.15. *For all $c \in Lc$:*

$$\{i \in Li | M(c,i)\} = \{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathcal{R}_{\mathfrak{C}}(c)\}.$$

Proof. The proof follows from definitions 2.7 and 2.14. □

We use the definition in the corollary above to determine when a concept language is complete. By theorem 2.16 below a concept language Lc is complete if and only if for each subset $I \subseteq Li$ there exists a concept description $c \in Lc$ such that the extensional representation of the concept given by c is equal to I.

Theorem 2.16. *A concept language Lc is complete with respect to the set $\mathfrak{C}$ of all possible concepts if and only if:*

$$(\forall I \subseteq Li)(\exists c \in Lc)(\{i \in Li | M(c,i)\} = I).$$

Proof.

$(\forall I \subseteq Li)(\exists c \in Lc)(\{i \in Li | M(c,i)\} = I)$ iff ($\mathcal{R}_{\mathfrak{I}}$ is bijective)
$(\forall \mathfrak{c} \in \mathfrak{C})(\exists c \in Lc)(\{i \in Li | M(c,i)\} = \{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}\})$ iff (definition 2.14)
$(\forall \mathfrak{c} \in \mathfrak{C})(\exists c \in Lc)(\{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathcal{R}_{\mathfrak{C}}(c)\} = \{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}\})$ iff (corollary 2.8)
$(\forall \mathfrak{c} \in \mathfrak{C})(\exists c \in Lc)(\mathcal{R}_{\mathfrak{C}}(c) = \mathfrak{c})$

But, according to definition 2.12 a concept language Lc is complete if and only if:

$$(\forall \mathfrak{c} \in \mathfrak{C})(\exists c \in Lc)(\mathcal{R}_{\mathfrak{C}}(c) = \mathfrak{c}).$$

Thus, the theorem is proven. □

In concept-learning tasks concepts under study are incompletely defined in instance languages. They are given with their sets of positive and negative training instances represented in instance languages[2].

Definition 2.17. (Set of Positive Training Instances) *A set $I^+ \subseteq Li$ is a set of descriptions of positive instances of a concept $\mathfrak{c} \in \mathfrak{C}$ if and only if:*

$$I^+ \subseteq \{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}\}.$$

Definition 2.18. (Set of Negative Training Instances) *A set $I^- \subseteq Li$ is a set of descriptions of negative instances of a concept $\mathfrak{c} \in \mathfrak{C}$ if and only if:*

$$I^- \cap \{i \in Li | \mathcal{R}_{\mathfrak{I}}(i) \in \mathfrak{c}\} = \emptyset.$$

Definition 2.17 states that an instance description $i \in I^+$ of a concept $\mathfrak{c} \in \mathfrak{C}$ represents in Li an instance $\mathfrak{i} \in \mathfrak{I}$ such that $\mathfrak{i} \in \mathfrak{c}$. Analogously, definition 2.18 states that an instance description $i \in I^-$ of a concept $\mathfrak{c} \in \mathfrak{C}$ represents in Li an instance $\mathfrak{i} \in \mathfrak{I}$ such that $\mathfrak{i} \notin \mathfrak{c}$.

From definitions 2.17 and 2.18 it is easy to see that the training sets I^+ and I^- are disjoint. This observation is shown in corollary 2.19 given below.

Corollary 2.19. *If a concept is given by the sets I^+ and I^- of positive and negative instances represented in an instance language Li then:*

$$I^+ \cap I^- = \emptyset.$$

To determine membership of instance descriptions to the training sets we introduce the function *label* in definition 2.20.

Definition 2.20. (Function *label*)
label : $Li \rightarrow \{+, -, ?\}$
defined by:

$$label(i) = \begin{cases} + & \text{if } i \in I^+; \\ - & \text{if } i \in I^-; \\ ? & \text{otherwise.} \end{cases}$$

[2] For the sake of brevity a set of training instances is called training set.

Given an instance description $i \in Li$ the function returns the label of the membership of the instance to the training sets[3]. If the instance i belongs to the set I^+ the label "+" is returned. If the instance i belongs to the set I^- the label "−" is returned. Otherwise, the label "?" is returned.

We restrict the training sets I^+ and I^- to be finite in constraint 2.21. The rationale behind the restriction is twofold. First, in practice we always have finite training sets. Second, the computational models of concept learning that we introduce after chapter 3 require finite training sets.

Constraint 2.21. *The training sets I^+ and I^- are finite.*

In the rest of the thesis we manipulate with concept instances represented in instance languages. Therefore, for the sake of simplicity we will use the term "instance" to refer to the term "description of instance".

2.2 Task Formalisation

We formalise the concept-learning task as a quadruple $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ in figure 2.1. Given an instance language Li, a concept language Lc, a membership relation M, and the training sets I^+ and I^- of a target concept, the task is to find descriptions of the concept in Lc that accurately classify instances in the language Li. A description $c \in Lc$ classifies accurately an instance i if and only if the fact that c does (not) cover i implies that i is (not) an instance of the target concept.

2.2.1 Consistency Criterion

Concept descriptions that accurately classify instances have to be specified by acceptance criteria. In the general case the acceptance criteria follow the fundamental assumption of inductive learning (Mitchell, 1997): *"Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples".*

One of the simplest criteria is the consistency criterion proposed in (Mitchell, 1978). The criterion is given as a function in definition 2.22 below.

Definition 2.22. (Consistency Criterion *cons***)**
$cons : Lc \times \langle \mathbf{P}(Li) \times \mathbf{P}(Li) \rangle \rightarrow Boolean$
defined by: $cons(c, \langle I_1, I_2 \rangle) \leftrightarrow ((\forall i \in I_1) M(c, i) \wedge (\forall i \in I_2) \neg M(c, i))$.

The definition states that a description $c \in Lc$ is consistent with a set I_1 considered as positive and a set I_2 considered as negative if and only if every instance $i \in I_1$ is covered by c, and every instance $i \in I_2$ is not covered by c. We use the

[3] The label of the membership of an instance to a training set is called training classification of the instance.

The Concept-Learning Task
$\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$.

Given:

- An instance language Li.
- A concept language Lc.
- A membership predicate M.
- Sets I^+ and I^- of positive and negative instances of a target concept.

Find: Descriptions of the target concept in Lc that accurately classify instances in Li.

Figure 2.1: The Concept-Learning Task.

consistency criterion to define consistent concept descriptions with respect to the training sets of a target concept.

Definition 2.23. (Consistent Concept Description) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$. *Then a description* $c \in Lc$ *is consistent with the sets* I^+ *and* I^- *if and only if* $cons(c, \langle I^+, I^- \rangle)$.

The concept-learning task can be specified by the consistency criterion. Hence, the task in this case is to *find descriptions in a concept language Lc that are consistent with the training sets of a target concept* (see figure 2.1).

2.2.2 Task Type

The concept-learning task can be *offline* and *online* (Langley, 1996). The task is *offline* when the training sets are known in advance. The task is *online* when training instances arrive sequentially in time. When instead of instances training subsets arrive sequentially in time, the task is intermediate; i.e., it has characteristics of both: offline and online tasks simultaneously.

The online task can be considered in the presence of *concept drift*. *Concept drift* holds when the meaning of target concepts is changed during online learning. This is revealed by a change in the training classification of some instances used.

The concept-learning task can be sophisticated when the training sets contain *noisy training instances. The noisy training instances are those that are incorrectly labelled with respect to target concepts.* We distinguish two classes of noisy instances: *false positive and false negative instances* (Mitchell, 1978). *The instances are false*

positive when they are incorrectly labelled as positive. Conversely, *the instances are false negative when they are incorrectly labelled as negative.*

2.3 Concept Learning as Search

Concept learning can be viewed as search in spaces defined by concept languages (Mitchell, 1978). The goal of the search is to find concept descriptions that satisfy a preliminary given acceptance criterion. The search spaces are usually structured according to a relation "more specific" that naturally occurs on most of the concept languages. The relation is defined to hold for any two descriptions c_1 and c_2 if and only if the extensional representation of c_1 is a subset of that of c_2. The search operators employ the "more specific" structures of the search spaces. For example, a specialisation operator finds concept descriptions that are more specific than concept descriptions given in its input. It can be realised for 1-CNF attribute languages by adding attributes to the initial concept descriptions. By duality, a generalisation operator finds concept descriptions that are less specific. It can be realised for 1-CNF attribute languages by removing attributes from the initial concept descriptions.

In addition to the search spaces and the operators a procedure is required that applies the operators to the search spaces. The procedure can be considered from four different angles: where to start the search, how to organise the search, how to evaluate alternative concept descriptions, and when to terminate the search. The search can start from the most specific, or the most general concept descriptions as well as from concept descriptions with intermediate generality. The search technique can be any method for exploring search spaces in the range of uninformed and informed search methods (Russel and Norvig, 1995). Evaluating alternative concept descriptions may employ some acceptance criteria. Terminating the search can be realised when it is not possible to find new concept descriptions that better fit the acceptance criteria used.

Analogously to the concept-learning task being online or offline, we distinguish incremental or nonincremental search procedures. A procedure is incremental if it processes sequentially training instances. Hence, it can be applied for online concept-learning tasks. Conversely, a procedure is nonincremental if it processes all training instances at once and thus it can be applied for offline concept-learning tasks.

2.4 Inductive Bias

The concept-learning task is ill-posed since its elements do not provide any basis for classifying instances that are not presented in the training sets (Mitchell, 1980; Utgoff, 1984). Hence, a concept learner needs a set of a priori assumptions about the identity of the concepts to be learned. This set is well-known as inductive bias. It is formalised by Mitchell (1997, p.63) as a *"set of assumptions that, together with the*

training data, deductively justify the classifications assigned by the learner to future instances".

The inductive bias is divided in two major categories: restriction bias and preference bias. A restriction bias assumes that the concept language is restricted (incomplete), and the unknown target concept is represented in the language. If no restrictions are imposed and a preference ordering is placed on the concept language, we have a preference bias.

2.5 Chapter Conclusion

In this chapter we have formalised the concept-learning task. We have shown that the task can be viewed as a search task. These findings will be elaborated in the rest of the thesis in the area of version spaces.

Chapter 3

Version-Space Abstract Data Type

This chapter deals with version spaces introduced in (Mitchell, 1978). We propose considering them as an abstract data type. The abstract data type is specified by the definition of version spaces in section 3.1, and four groups of their basic operations, introduced in section 3.2. Since version-space learning can be viewed as search, the first group consists of search operations: *Initialise*, *Update*, and *Retraction*. They determine how to initialise and how to transform version spaces when instances are added to or removed from the training sets. The status of the search can be identified by the second group of state-test operations *Collapsed?* and *Converged?*. The operations determine when version spaces are empty and when they consist of equivalent concept descriptions. The third group consists of only one operation *Classify*. The operation classifies instances by the rule of the unanimous vote of the descriptions in version spaces (Mitchell, 1978). Since version spaces are sets, the last (fourth) group consists of set operations *Member?*, *Intersection*, *Subset?*, and *Equal?*.

Following Mitchell (1997) we give some remarks on version spaces in section 3.3. We discuss how to use them for selecting new training instances in subsection 3.3.1. We analyse the conditions when version spaces contain correct concept descriptions in subsection 3.3.2. The analysis is used in order to determine two principal problems of version spaces, namely the problem with incomplete concept languages and the problem with noisy training instances that correspond to the first and fourth part of our problem statement.

In section 3.4 the inductive bias of version spaces is informally considered.

The chapter ends with some basic conclusions in section 3.5 that determine the main lines of research presented in the next chapters of the thesis.

3.1 Definition of Version Spaces

Version spaces are sets of all possible descriptions in concept languages that are consistent with positive and negative training instances of target concepts. Hence, version spaces are complete solutions of concept-learning tasks specified by the consistency criterion (Mitchell, 1978). More formally:

Definition 3.1. (Version Space (VS)) *The version space VS of a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *is defined as follows:*

$$VS = \{c \in Lc \,|\, cons(c, \langle I^+, I^- \rangle)\}.$$

We consider version spaces as an abstract data type (Hirsh, 1992b). This requires specifying their set of basic operations in addition to their model in definition 3.1 (Goodrich and Tamassia, 1998); we do so in section 3.2.

3.2 Basic Version-Space Operations

The basic version-space operations are taken from (Mitchell, 1978; Idemstam-Almquist, 1990; Hirsh, 1992b; Hirsh *et al.*, 1997). We define them as functions. The operations are divided into four subgroups: search operations, state-test operations, classification operation, and additional set operations.

3.2.1 Search Operations

In the previous chapter we have considered concept learning as search. This viewpoint holds for version spaces (Mitchell, 1978). The search space is the set of all possible version spaces defined in the concept languages used. The search operators are given by the operations: *Initialise*, *Update*, and *Retraction* considered in the next subsections.

3.2.1.1 Operation *Initialise*

To start the search we have to know where to start from. When no training instances of a target concept are available, the version space is set equal to the concept language used (see corollary 3.2).

Corollary 3.2. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space VS. If* $I^+ = \emptyset$ *and* $I^- = \emptyset$ *then* $VS = Lc$.

When preliminary information about the identity of the target concept is available the concept language can be restricted according to that information. Hence, the version space can be set equal to the restricted language.

To comprise both cases described above we specify the operation *Initialise* as a function in definition 3.3 below.

Definition 3.3. (Operation *Initialise*)
Initialise : $\mathbf{P}(Lc) \rightarrow \mathbf{P}(Lc)$[1]
defined by: $Initialise(L) = L$.

3.2.1.2 Operation *Update*

The operation *Update* transforms the version space of a target concept whenever an instance is added to the training sets (Mitchell, 1978; Hirsh, 1992b; Hirsh *et al.*, 1997). It is based on theorems 3.4 and 3.5 given below. The first theorem determines a procedure for transforming version spaces given a new positive training instance; the second theorem determines a procedure for transforming version spaces given a new negative training instance.

Theorem 3.4. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space* VS*; and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with version space* VS'*, where* $I^{+\prime} = I^+ \cup \{i\}$*. Then the version spaces* VS *and* VS' *obey the following equality:*

$$VS' = \{c \in VS | M(c, i)\}.$$

Proof. (The proof is taken from (Mitchell, 1978)) According to definition 2.22 we have that:

$$cons(c, \langle I^{+\prime}, I^- \rangle) \leftrightarrow (cons(c, \langle I^+, I^- \rangle) \wedge M(c, i)).$$

Thus, using definition 3.1 it follows that:

$$\begin{aligned} VS' &= \{c \in Lc | cons(c, \langle I^{+\prime}, I^- \rangle)\} \\ &= \{c \in Lc | cons(c, \langle I^+, I^- \rangle) \wedge M(c, i)\} \end{aligned}$$

But, the version space VS of the task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ is given as follows:

$$VS = \{c \in Lc | cons(c, \langle I^+, I^- \rangle)\}.$$

Thus,

$$\begin{aligned} VS' &= \{c \in Lc | (c \in VS) \wedge M(c, i)\} \\ &= \{c \in VS | M(c, i)\} \end{aligned}$$

□

[1] $\mathbf{P}(S)$ denotes the power-set of a set S.

Theorem 3.5. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space* VS*; and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle\rangle$ *with version space* VS'*, where* $I^{-\prime} = I^- \cup \{i\}$*. Then the version spaces* VS *and* VS' *obey the following equality:*

$$VS' = \{c \in VS | \neg M(c, i)\}.$$

Proof. The proof is dual to that of theorem 3.4. □

The theorems above determine the operation *Update*. Its input is the version space VS of a target concept. If a new positive instance $i \in Li$ is given, the operation updates the version space VS according to theorem 3.4. Thus, the new updated version space VS' is formed from those elements of VS that do cover the instance i.

Conversely, if a new negative instance $i \in Li$ is given, the operation updates the version space VS according to theorem 3.5. Thus, the new updated version space VS' is formed from those elements of VS that do not cover the instance i.

The operation *Update* is defined as a function. Its definition is given below.

Definition 3.6. (Operation *Update***)**
Update : $Li \times \mathbf{P}(Lc) \rightarrow \mathbf{P}(Lc)$
defined by:

$$Update(i, VS) = \begin{cases} \{c \in VS | M(c, i)\} & \text{if } i \text{ is positive;} \\ \{c \in VS | \neg M(c, i)\} & \text{if } i \text{ is negative.} \end{cases}$$

3.2.1.3 Operation *Retraction*

The operation *Retraction* transforms the version space of a target concept whenever an instance is removed from the training sets (Idemstam-Almquist, 1990; Hirsh *et al.*, 1997). It is based on theorems 3.7 and 3.8 given below. The first theorem determines a procedure for transforming version spaces when a positive training instance is retracted; the second theorem determines a procedure for transforming version spaces when a negative training instance is retracted.

Theorem 3.7. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space* VS*; a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with version space* VS'*, where* $I^{+\prime} = I^+ - \{i\}$ *and* $i \in I^+$*. Then the version spaces* VS *and* VS' *obey the following equality:*

$$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \cup \{i\}\rangle)\}.$$

Proof.

$$\begin{aligned} VS' &= \{c \in Lc | cons(c, \langle I^{+\prime}, I^- \rangle)\} \\ &= \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \rangle)\} \\ &= \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge (M(c,i) \vee \neg M(c,i))\} \\ &= \{c \in Lc | (cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge M(c,i)) \vee \\ &\qquad (cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge \neg M(c,i))\} \\ &= \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge M(c,i)\} \cup \\ &\quad \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge \neg M(c,i)\} \end{aligned}$$

According to definition 2.22:

$$cons(c, \langle I^+, I^- \rangle) \leftrightarrow (cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge M(c,i)).$$
$$cons(c, \langle I^+ - \{i\}, I^- \cup \{i\} \rangle) \leftrightarrow (cons(c, \langle I^+ - \{i\}, I^- \rangle) \wedge \neg M(c,i)).$$

Thus,

$$\begin{aligned} VS' &= \{c \in Lc | cons(c, \langle I^+, I^- \rangle)\} \cup \\ &\quad \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \cup \{i\} \rangle)\} \end{aligned}$$

But, the version space VS of the task $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ is given as follows:

$$VS = \{c \in Lc | cons(c, \langle I^+, I^- \rangle)\}.$$

Thus,

$$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \cup \{i\} \rangle)\}.$$

□

Theorem 3.8. *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ with version space VS; and a second task $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle \rangle$ with version space VS', where $I^{-\prime} = I^- - \{i\}$ and $i \in I^-$. Then the version spaces VS and VS' obey the following equality:*

$$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ \cup \{i\}, I^- - \{i\} \rangle)\}.$$

Proof. The proof is dual to that of theorem 3.7. □

The theorems determine the operation *Retraction*. Its input is the version space VS of a target concept. If a positive instance $i \in Li$ is removed from the training set I^+, then the concept-learning task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ is changed to a new one $\langle Li, Lc, M, \langle I^+ - \{i\}, I^- \rangle\rangle$ and the operation functions according to theorem 3.7. Thus, the operation forms the version space VS' of the new task as union of the version space VS of the old task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ and a version space $\{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \cup \{i\}\rangle)\}$.

Conversely, if a negative instance $i \in Li$ is removed from the training set I^-, then the concept-learning task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ is changed to a new one $\langle Li, Lc, M, \langle I^+, I^- - \{i\}\rangle\rangle$ and the operation functions according to theorem 3.8. Thus, the operation forms the version space VS' of the new task as union of the version space VS of the old task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ and a version space $\{c \in Lc | cons(c, \langle I^+ \cup \{i\}, I^- - \{i\}\rangle)\}$.

The operation *Retraction* is defined as a function. Its definition is given below.

Definition 3.9. (Operation *Retraction*)
Retraction : $Li \times \mathbf{P}(Lc) \times \mathbf{P}(Lc) \times \mathbf{P}(Li) \times \mathbf{P}(Li) \rightarrow \mathbf{P}(Lc)$
defined by: $Retraction(i, VS, Lc, I^+, I^-) =$

$$\begin{cases} VS \cup \{c \in Lc | cons(c, \langle I^+ - \{i\}, I^- \cup \{i\}\rangle)\} & \text{if } i \in I^+; \\ VS \cup \{c \in Lc | cons(c, \langle I^+ \cup \{i\}, I^- - \{i\}\rangle)\} & \text{if } i \in I^-. \end{cases}$$

where: $VS = \{c \in Lc | cons(c, \langle I^+, I^- \rangle)\}$.

3.2.2 State-Test Operations

The state-test operations determine the state of the version space of a target concept. We distinguish two state-test operations *Collapsed?* and *Converged?* presented in the next two subsections.

3.2.2.1 Operation *Collapsed?*

The operation *Collapsed?* determines when a version space is empty (collapsed). It is defined as a function below.

Definition 3.10. (Operation *Collapsed?*)
Collapsed? : $\mathbf{P}(Lc) \rightarrow Boolean$
defined by: $Collapsed?(VS)$ *iff* $VS = \emptyset$.

A version space is empty when its target concept is not represented in the concept language. This can be due to deficiencies in the concept language, the instance language, and the membership relation as well as noisy training instances.

3.2.2.2 Operation *Converged?*

The operation *Converged?* determines the desired state of a version space consisting of equivalent concept descriptions only. Concept descriptions are equivalent if and only if an equivalence relation holds. The relation is specified in definition 3.11.

Definition 3.11. (Equivalence Relation on Lc ($\sim$)) *Consider an instance language Li, a concept language Lc, and a membership relation M. Then:*

$$(\forall c_1, c_2 \in Lc)((c_1 \sim c_2) \leftrightarrow (\forall i \in Li)(M(c_1, i) \leftrightarrow M(c_2, i))).$$

The definition states that two concept descriptions $c_1, c_2 \in Lc$ are equivalent $(c_1 \sim c_2)$ if and only if the equivalence $M(c_1, i) \leftrightarrow M(c_2, i)$ holds for all instances $i \in Li$. In other words, by corollary 2.15 the descriptions are equivalent if and only if the extensional representations of the concepts, that they stand for, are equal.

We use the notion of equivalent concept descriptions to define the operation *Converged?* as a function below.

Definition 3.12. (Operation *Converged?*)
Converged? : $\mathbf{P}(Lc) \rightarrow Boolean$
defined by: Converged?(VS) *iff* $((VS \neq \emptyset) \wedge (\forall c_1, c_2 \in VS)(c_1 \sim c_2))$.

According to definition 3.12 in order to determine whether a version space VS is converged we have to test VS for a collapse. Hence, the algorithms of the operation *Converged?* have to use the operation *Collapsed?*.

Note that when the function $\mathcal{R}_{\mathfrak{C}}$ is injective, concept descriptions are equivalent if and only if they are equal. This property is proven in lemma 3.13 given below.

Lemma 3.13. *Consider an instance language Li, a concept language Lc, and a membership relation M. If $\mathcal{R}_{\mathfrak{C}}$ is injective then:*

$$(\forall c_1, c_2 \in Lc)((c_1 \sim c_2) \leftrightarrow (c_1 = c_2)).$$

Proof. ($\rightarrow$) Consider arbitrarily $i \in Li$ and $c_1, c_2 \in Lc$ such that $c_1 \sim c_2$. According to definition 3.11 $M(c_1, i) \leftrightarrow M(c_2, i)$. Thus, using definition 2.14:

$$\mathcal{R}_{\mathfrak{I}}(i) \in \mathcal{R}_{\mathfrak{C}}(c_1) \leftrightarrow \mathcal{R}_{\mathfrak{I}}(i) \in \mathcal{R}_{\mathfrak{C}}(c_2).$$

This implies:

$$\mathcal{R}_{\mathfrak{C}}(c_1) = \mathcal{R}_{\mathfrak{C}}(c_2).$$

Since $\mathcal{R}_{\mathfrak{C}}$ is injective:

$$c_1 = c_2.$$

The first part of the theorem is proven.

($\leftarrow$) The second part of the proof follows immediately from definition 3.11. □

In chapter 2 we have assumed that the function $\mathcal{R}_{\mathfrak{C}}$ is injective. Therefore, by lemma 3.13 a version space consists of equivalent concept descriptions if and only if it has exactly one concept description. This implies that *the operation Converged? returns true if and only if the version space has exactly one concept description.*

Note that if a version space consists of only one concept description, then this is an indication that the target concept is presented in the concept language used and it is uniquely specified. In general, this requires a very large number of training instances. The smaller the number of non-equivalent concept descriptions is, the longer we have to wait for a random instance that can distinguish them (Haussler, 1988).

When both operations *Collapsed?* and *Converged?* fail, it is an indication that the version space contains non-equivalent concept descriptions. This means that the target concept is presented in the concept language but it is not uniquely specified.

The state-test operations can be used in order to determine when to stop or direct the process of the version space search. For example, if the operation *Converged?* succeeds and no additional training instances are available, then the search is successful and it stops. Conversely, if the operation *Collapsed?* succeeds and we have an information that some of the training instances are noisy, then we can retract them. This means that we redirect the search process and can continue with new training instances.

3.2.3 Operation *Classify*

The operation *Classify* determines the membership of an instance to a target concept given the version space of the concept. It is formally specified as a function in definition 3.14 given below.

Definition 3.14. (Operation *Classify*)
Classify : $Li \times \mathbf{P}(Lc) \rightarrow \{+, -, ?\}$
defined by:

$$Classify(i, VS) = \begin{cases} + & \text{if } (VS \neq \emptyset) \wedge (\forall c \in VS) M(c, i); \\ - & \text{if } (VS \neq \emptyset) \wedge (\forall c \in VS) \neg M(c, i); \\ ? & \text{otherwise.} \end{cases}$$

The operation *Classify* realises the rule of the unanimous vote of descriptions in version spaces (Mitchell, 1978; Hirsh, 1992b). Given the version space VS of a target concept it functions as follows:

(1) if VS is nonempty and all the descriptions in VS cover an instance $i \in Li$, then the operation classifies the instance i as positive with respect to the target concept and returns "+";

(2) if VS is nonempty and all the descriptions in VS do not cover an instance $i \in Li$, then the operation classifies the instance i as negative with respect to the target concept and returns "$-$";

(3) if (1) and (2) do not hold, then the operation cannot classify an instance $i \in Li$ and returns "?".

According to definition 3.14 algorithms of the operation *Classify* have to use the operation *Collapsed?*. This is due to the fact that they have to check whether the version space, used in classification, is empty.

Note that nonempty version spaces can be used for classification in two possible states. The first one is when they are converged. In this case the classification of the instances is unambiguously determined. The second state is when they are not converged. In this case the classification of the instances is not guaranteed. Nevertheless, the fact that version spaces can be used for classification, when target concepts are not uniquely specified, is considered as one of the main sources of the power of version spaces.

3.2.3.1 Implementation

The operation *Classify* can be implemented using the operations *Update* and *Collapsed?*. The implementation is based on corollaries 3.15 and 3.16 given below. Corollary 3.15 states that all the descriptions of a version space VS do cover an instance $i \in Li$ if and only if the subset of VS which descriptions do not cover i is empty. By duality, corollary 3.16 states that all the descriptions of a version space VS do not cover an instance $i \in Li$ if and only if the subset of VS which descriptions do cover i is empty.

Corollary 3.15. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space* VS. *Then:*

$$(\forall i \in Li)((\forall c \in VS)M(c,i) \leftrightarrow \{c \in VS | \neg M(c,i)\} = \emptyset).$$

Corollary 3.16. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space* VS. *Then:*

$$(\forall i \in Li)((\forall c \in VS)\neg M(c,i) \leftrightarrow \{c \in VS | M(c,i)\} = \emptyset).$$

By corollaries 3.15 and 3.16 the operation *Classify* can be implemented with an algorithm based on the operations *Update* and *Collapsed?*. When an instance has to be classified then the algorithm updates the version space of a target concept with the instance considered as negative. If the version space collapses, then according to definition 3.6 and corollary 3.15 all the descriptions in the version space cover

the instance. Hence, the instance classification is positive for the target concept. If the version space does not collapse, then the algorithm updates the original version space with the instance considered as positive. If the version space collapses, then according to definition 3.6 and corollary 3.16 all the descriptions in the version space do not cover the instance. Hence, the instance classification is negative for the target concept. If after both updates the version space does not collapse, then this is an indication that some of the descriptions in version space cover the instance and some do not. Hence, the instance classification cannot be determined.

3.2.3.2 Influence of the Concept-Language Completeness

The operation *Classify* can be used for classifying instances, that are not presented in the training sets, only when the concept languages used are not complete. In order to understand the reason we consider again the notion of completeness. By theorem 2.16 a concept language Lc is complete if and only if for each subset I in the instance language Li there exists a concept description $c \in Lc$ such that the extensional representation of the concept given by c is equal to I. Using definition 2.22 we formalise this result with the consistency predicate. Thus, a concept language Lc is complete if and only if [2]:

$$(\forall I \subseteq Li)(\exists c \in Lc) cons(c, \langle I, Li - I \rangle).$$

We use this new formalisation of theorem 2.16 in corollary 3.17. The corollary states that the version spaces for all possible tasks are non-empty if and only if the concept language is complete.

Corollary 3.17. *Consider an instance language Li, a concept language Lc, and a membership relation M. Then the version spaces VS for all possible tasks $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ are not empty if and only if:*

$$(\forall I \subseteq Li)(\exists c \in Lc) cons(c, \langle I, Li - I \rangle).$$

The corollary can be used in order to explain the classification process based on the rule of the unanimous vote of the descriptions in version spaces. Consider a non-empty version space VS, defined in a concept language Lc by arbitrarily chosen training sets I^+ and I^-, and an arbitrary instance $i \in Li - (I^+ \cup I^-)$. To classify the instance as positive we have to show that $(\forall c \in VS) M(c, i)$. By corollary 3.15 it is equivalent to show that the set $\{c \in VS | \neg M(c, i)\}$ is empty. Note that by theorem 3.4 the set is the version space of the task $\langle Li, Lc, M, \langle I^+, I^- \cup \{i\}\rangle\rangle$. Therefore,

(1) if the concept language Lc is complete, then by corollary 3.17 the version space $\{c \in VS | \neg M(c, i)\}$ is not empty. This means that the instance i is not covered by all the descriptions in the version space VS and its positive classification cannot be determined.

[2] Note that in the rest of the thesis theorem 2.16 will be considered only in this form.

(2) if the concept language Lc is not complete, then by corollary 3.17 there exists at least one empty version space in Lc. Therefore, if this version space is exactly the version space $\{c \in VS | \neg M(c, i)\}$ then the instance i is covered by all the descriptions in the version space VS and it is positive.

Since the version space VS is arbitrary, results (1) and (2) hold for all possible version spaces defined in the concept language Lc. By analogy, similar derivations can be made when instances have to be classified as negative. Thus, we conclude that when concept languages are complete, version spaces cannot be used for classification of unseen instances using the rule of the unanimous vote of their descriptions.

3.2.4 Additional Set Operations

Since version spaces are sets, we introduce four additional set operations. The operations were proposed in (Hirsh, 1992b; Hirsh *et al.*, 1997) and are given below[3].

3.2.4.1 Operation *Member?*

The operation *Member?* determines whether a description in the concept language belongs to the version space of a target concept. It is defined as a function given below.

Definition 3.18. (Operation *Member?*)
Member? : $Lc \times \mathbf{P}(Lc) \rightarrow Boolean$
defined by: Member?(c, VS) *iff* $c \in VS$.

3.2.4.2 Operation *Intersection*

The operation *Intersection* computes a version space that is the intersection of two other version spaces. It is defined as a function given below.

Definition 3.19. (Operation *Intersection*)
Intersection : $\mathbf{P}(Lc) \times \mathbf{P}(Lc) \rightarrow \mathbf{P}(Lc)$
defined by: Intersection$(VS_1, VS_2) = VS_1 \cap VS_2$.

The operation is closely related to theorem 3.20 from (Hirsh, 1989). The theorem states that a version space, based on the training instances of two other version spaces, is equal to the intersection of these version spaces. That is why, Hirsh (1994) used the intersection operation as a basis for an alternative approach to version space learning.

[3] Note that these operations do not exhaust all the set operations. This is due to the fact that some of the set operations are meaningless for version spaces. For example, the union or the difference of two version spaces is not always a version space in spite of the fact that it is a set (Hirsh *et al.*, 1997).

Theorem 3.20. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with version space* VS_1*, a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2*, and a third task* $\langle Li, Lc, M, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^- \rangle\rangle$ *with version space* VS_{12}*. Then:*

$$VS_{12} = VS_1 \cap VS_2.$$

Proof. The proof is done by first showing that $VS_{12} \subseteq VS_1 \cap VS_2$, then showing that $VS_{12} \supseteq VS_1 \cap VS_2$.

($\subseteq$) Consider an arbitrary chosen $c \in VS_{12}$. According to definition 3.1 it follows that $cons(c, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^- \rangle)$. Using definition 2.22 the consistency predicate implies:

$$(\forall i \in I_1^+) M(c, i) \tag{3.1}$$
$$(\forall i \in I_2^+) M(c, i) \tag{3.2}$$
$$(\forall i \in I_1^-) \neg M(c, i) \tag{3.3}$$
$$(\forall i \in I_2^-) \neg M(c, i) \tag{3.4}$$

From (3.1) and (3.3) using definition 2.22 it follows that $cons(c, \langle I_1^+, I_1^- \rangle)$. According to definition 3.1 $cons(c, \langle I_1^+, I_1^- \rangle)$ implies that $c \in VS_1$.
Analogously, from (3.2) and (3.4) using definition 2.22 it follows that $cons(c, \langle I_2^+, I_2^- \rangle)$. According to definition 3.1 $cons(c, \langle I_2^+, I_2^- \rangle)$ implies that $c \in VS_2$.

($\supseteq$) Consider an arbitrary chosen $c \in Lc$ such that $c \in VS_1$ and $c \in VS_2$. Thus, according to definition 3.1 we have that $cons(c, \langle I_1^+, I_1^- \rangle)$ and $cons(c, \langle I_2^+, I_2^- \rangle)$. Using definition 2.22 the consistency predicates imply again formulas (3.1) to (3.4), and thus it follows that $cons(c, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^- \rangle)$. According to definition 3.1 the last dependency implies that $c \in VS_{12}$. □

3.2.4.3 Operation *Subset?*

The operation *Subset?* determines whether a version space is a subset of another version space. Its function is formally specified in definition 3.21 given below.

Definition 3.21. (Operation *Subset?*)
Subset? : $\mathbf{P}(Lc) \times \mathbf{P}(Lc) \rightarrow$ *Boolean*
defined by: Subset?(VS_1, VS_2) *iff* $VS_1 \subseteq VS_2$.

For some concept languages implementations of the operation can be based on theorem 3.22 below. The theorem states that a version space VS_1 is a subset of a version space VS_2 based on training sets I_2^+ and I_2^- if and only if every positive instance $i \in I_2^+$ is covered by all the descriptions in VS_1, and every negative instance $i \in I_2^-$ is not covered by any description in VS_1.

Theorem 3.22. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with version space* VS_1*, and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2*. Then:*

$$VS_1 \subseteq VS_2 \leftrightarrow ((\forall c \in VS_1)(\forall i \in I_2^+)M(c,i) \wedge (\forall c \in VS_1)(\forall i \in I_2^-)\neg M(c,i)).$$

Proof.

$VS_1 \subseteq VS_2$ iff
$(\forall c \in VS_1)(c \in VS_2)$ iff (definition 3.1)
$(\forall c \in VS_1)cons(c, \langle I_2^+, I_2^- \rangle)$ iff (definition 2.22)
$(\forall c \in VS_1)((\forall i \in I_2^+)M(c,i) \wedge (\forall i \in I_2^-)\neg M(c,i))$ iff
$(\forall c \in VS_1)(\forall i \in I_2^+)M(c,i) \wedge (\forall c \in VS_1)(\forall i \in I_2^-)\neg M(c,i)$

□

3.2.4.4 Operation *Equal?*

The operation *Equal?* determines whether a version space is equal to another version space. It is defined as a function given below.

Definition 3.23. (Operation *Equal?*)
Equal? : $\mathbf{P}(Lc) \times \mathbf{P}(Lc) \rightarrow Boolean$
defined by: Equal?(VS_1, VS_2) *iff* $VS_1 = VS_2$.

Implementations of the operation *Equal?* can be based on the operation *Subset?*. This is due to a definition of the set equality given below:

$$VS_1 = VS_2 \leftrightarrow (VS_1 \subseteq VS_2 \wedge VS_2 \subseteq VS_1).$$

Thus, in order to determine whether a version space VS_1 is equal to a version space VS_2 we have to check whether VS_1 is a subset of VS_2 and VS_2 is a subset of VS_1. If so, by the set equality definition both version spaces are equal. Otherwise, both version spaces are not equal.

3.3 Remarks on Version Spaces

3.3.1 Selecting New Training Instances

Version spaces contain information for selecting new training instances. Consider an instance that is covered by half of the descriptions in a version space. Independently of its actual classification, when it is given, the operation *Update* will reduce the size of the version space by a factor 2 according to theorems 3.4 and 3.5 (Mitchell, 1978). Therefore, the target concept can be identified with a sequence of $\lceil log_2(|VS|)\rceil$ training instances. Unfortunately, the problem of finding such a sequence is in general *NP*-hard (Subramanian and Feigenbaum, 1986).

3.3.2 Version Spaces and Correct Concept Descriptions

Mitchell (1997) has shown that a version space contains the correct description of a target concept if:

(1) the concept language used includes this description; and

(2) the training instances are noise free.

Condition (1) follows from definition 3.1 of version spaces. If it holds, we say that the completeness of the concept language is sufficient with respect to the concept-learning task to be solved. Otherwise, we say that the completeness is not sufficient and we face the problem of incomplete concept languages that corresponds to the first part of our problem statement

Condition (2) follows from definition 3.6 of the operation *Update*. Assume that we have a new training instance that is noisy. If we apply the operation *Update*, then we remove descriptions from version spaces that are inconsistent with the instance. Since the classification of the instance is not correct, this means that we remove the description of the target concept. Therefore, we face the problem of noisy training instances that corresponds to the fourth part of our problem statement.

3.4 Inductive Bias

The inductive bias of version spaces is the assumption that the description c of the target concept is in the concept language (Mitchell, 1997). Note that the description c has to cover all positive training instances and has to reject all negative training instances. This means that c is consistent with the training data. Thus, by definition 3.1 c has to belong to the version space VS of the target concept. Since $c \in VS$ the positive or negative classification that any instance i can obtain by the operation *Classify* is equal to the classification that the description c assigns to the instance i. Note that the operation can classify unambiguously if the concept language is not complete; i.e., restricted (see section 3.2.3.2). Thus, the inductive bias of version spaces is a restriction bias.

3.5 Chapter Conclusion

In this chapter we have considered version spaces proposed by Mitchell (1978). We have proposed considering them as an abstract data type. The type has been completely specified by the definition of version spaces in section 3.1 and their set of basic operations in section 3.2. Most of the operations were proposed by Mitchell (1978). The operation *Retraction* was introduced by Idemstam-Almquist (1990). Hirsh (1992b) added the operations *Subset?* and *Equal?* and proposed to consider version spaces in terms of their operations. A shortcoming of Hirsh's consideration

was the informal description of the operations. Therefore, *the main contribution of this chapter is the rigorous specification of the version-space abstract data type.*

The abstract data type gives some useful insights into version spaces. For example, all the search operations are incremental since they process training instances sequentially in time. If we compare the operations *Update* and *Retraction*, then we shall find that the former does not require storing training instances while the latter does. Hence, we can say that the operation *Update* is more strongly incremental than the operation *Retraction*. A deeper analysis of the operations shows that they are instance-order independent. This means that different orders of the same training sets imply the same version spaces. This nice property follows from theorems 3.4 and 3.5 for the operation *Update*, and from theorems 3.7 and 3.8 for the operation *Retraction*.

In addition to the positive side of version spaces the chapter has revealed their two well-known principal problems that correspond to the first and fourth part of our problem statement. More precisely,

- the problem with incomplete concept languages; i.e., the inability to learn when the completeness of the concept languages used is not sufficient with respect to the concept-learning tasks to be solved; and
- the problem with noisy training instances; i.e., the inability to learn correct concept descriptions when at least one training instance is noisy.

In chapters 4 and 7 we propose a partial solution to the first problem. The core idea is to define version spaces in the conjunctive or disjunctive extensions of concept languages. We shall prove that version spaces defined in this way are more complete than ordinary version spaces and that is why they are solutions of more concept-learning tasks. In chapter 8 we propose a solution to the second problem. The core idea is to apply preference biases on version spaces that are capable of handling noise in the training instances.

Chapter 4

Conjunctive and Disjunctive Version Spaces

In this chapter we try to obtain more insights into the first part of our problem statement, namely the problem with incomplete concept languages. The core idea is to extend concept languages with logical conjunction when their completeness is not sufficient with respect to the concept-learning tasks to be solved. Conjunctive extensions of concept languages are defined and their completeness properties are analysed in section 4.1. We prove that they are more complete than ordinary concept languages. This is the main motivation of the research, proposed in the chapter: if a target concept is not presented in concept languages, then it is possible that the concept is presented in their conjunctive extensions.

Conjunctive version spaces are introduced in section 4.2. We define them as version spaces in conjunctive extensions of concept languages. Their completeness property are analysed. We prove that conjunctive version spaces are more complete than (ordinary) version spaces. Hence, they are solutions of more concept learning tasks. In this context we show that conjunctive version spaces are a partial solution to the problem of incomplete concept languages.

Following chapter 3 we show that conjunctive version spaces can be considered as an abstract data type. We prove that the conjunctive version-space abstract data type can be expressed in terms of the version-space abstract data type in section 4.3. More precisely:

- conjunctive version spaces can be represented by version spaces defined in ordinary concept languages (see subsection 4.3.1); and
- most of the basic conjunctive version-space operations can be represented by algorithms which main operators are the basic version-space operations (see subsection 4.3.2).

By duality we can define disjunctive extensions of concept languages, disjunctive version spaces, and their abstract data type. We skip this part of the chapter since it is analogous to the part of conjunctive version spaces.

The chapter ends with some basic conclusions that determine the main lines of research presented in the next chapters of the thesis.

4.1 Conjunctive Extensions of Concept Languages

4.1.1 Definition

To define conjunctive extensions of concept languages we introduce the notion of conjunction.

Definition 4.1. (Conjunction) *Consider a concept language Lc. Then every non-empty subset $C \subseteq Lc$ is a conjunction and the descriptions $c \in C$ are conjuncts.*

Notation 4.2. *A conjunction C is denoted as $\bigwedge_{c \in C} c$.*

According to definition 4.1 a conjunction C is an element of $\mathbf{P_1}(Lc)$[1]. We use this observation to define conjunctive extensions of concept languages.

Definition 4.3. (Conjunctive Extension of Concept Languages) *Consider a concept language Lc. Then CLc is a conjunctive extension of Lc if and only if:*

$$CLc = \mathbf{P_1}(Lc).$$

According to definition 4.3 the conjunctive extension CLc of a concept language Lc is equal to the power-set of Lc that does not contain the empty set. The elements of CLc are considered as conjunctions. Thus, if the concept language Lc is finite, then the conjunctions $C \in CLc$ are finite. Conversely, if the concept language Lc is infinite, then the conjunctions $C \in CLc$ can be both finite or infinite.

To relate conjunctive extensions with instance languages we extend the membership relation in definition 4.4 given below.

Definition 4.4. (Conjunctive Membership Relation M_C)
$M_C : CLc \times Li \rightarrow Boolean$
defined by: $M_C(C, i) \leftrightarrow (\forall c \in C) M(c, i)$.

Definition 4.4 states that the conjunctive membership relation $M_C(C, i)$ holds for a conjunction $C \in CLc$ and an instance $i \in Li$ if and only if the relation $M(c, i)$ holds for all conjuncts $c \in C$. Conversely, the relation $M_C(C, i)$ does not hold for a

[1] $\mathbf{P_1}(S)$ denotes the powerset of a set S that does not include the empty set.

conjunction $C \in CLc$ and an instance $i \in Li$ if and only if the relation $M(c, i)$ does not hold for at least one conjunct $c \in C$.

We say that a conjunction $C \in CLc$ covers an instance $i \in Li$ if and only if $M_C(C, i)$ holds. Conversely, we say that a conjunction $C \in CLc$ does not cover (rejects) an instance $i \in Li$ if and only if $M_C(C, i)$ does not hold.

4.1.2 Conjunctive Consistency Criterion

Since the membership relation has been changed, we specify in definition 4.5 the consistency criterion for the conjunctive case.

Definition 4.5. (Conjunctive Consistency Criterion $cons_C$)
$cons_C : CLc \times \langle \mathbf{P}(Li) \times \mathbf{P}(Li) \rangle \rightarrow Boolean$
defined by: $cons_C(C, \langle I_1, I_2 \rangle) \leftrightarrow ((\forall i \in I_1) M_C(C, i) \wedge (\forall i \in I_2) \neg M_C(C, i))$.

We use the conjunctive consistency criterion to define the notion of consistent conjunctions in definition 4.6 given below.

Definition 4.6. (Consistent Conjunctions) *A conjunction $C \in CLc$ is consistent with sets I^+ and I^- if and only if $cons_C(C, \langle I^+, I^- \rangle)$.*

Definition 4.6 states that a conjunction $C \in CLc$ is consistent with the sets I^+ and I^- if and only if every instance $i \in I^+$ is covered by C, and every instance $i \in I^-$ is not covered by C.

4.1.3 Completeness

Conjunctive extensions of concept languages have been introduced because they are more complete than the languages themselves. This property is proven in theorem 4.7 given below. The theorem states that if a concept, extensionally represented by a set I in an instance language Li, is intensionally represented in a concept language Lc, then the concept is intensionally represented in the conjunctive extension CLc of Lc. Note that the other way around is in general not true. Therefore, all concepts, represented in concept languages, are represented in the conjunctive extensions of these languages.

Theorem 4.7. *Consider a concept language Lc and its conjunctive extension CLc. Then:*

$$(\forall I \subseteq Li)((\exists c \in Lc) cons(c, \langle I, Li - I \rangle) \rightarrow (\exists C \in CLc) cons_C(C, \langle I, Li - I \rangle)).$$

Proof. Consider an arbitrarily chosen set $I \subseteq Li$ such that there exists $c \in Lc$ and $cons(c, \langle I, Li - I \rangle)$. Thus, using definition 2.22 we have that:

$$(\forall i \in I)M(c,i) \tag{4.1}$$
$$(\forall i \in Li - I)\neg M(c,i). \tag{4.2}$$

According to definitions 4.1 and 4.3 the set $C = \{c\}$ is a conjunction and $C \in CLc$. Using definition 4.4 formula (4.1) implies $(\forall i \in I)M_C(C,i)$ and formula (4.2) implies $(\forall i \in Li - I)\neg M_C(C,i)$. According to definition 4.5 the last two derivations imply $cons_C(C, \langle I, Li - I\rangle)$. Thus,

$$(\exists C \in CLc)cons_C(C, \langle I, Li - I\rangle).$$

□

Since conjunctive extensions *CLc* are more complete than their corresponding concept languages *Lc*, it is reasonable to determine when they are fully complete. By theorem 2.16 conjunctive extensions *CLc* are complete if and only if for each subset I in the instance language *Li* there exists a conjunction $C \in CLc$ such that the extensional representation of the concept given by C is equal to I. Since conjunctive extensions *CLc* are defined on concept languages *Lc*, we determine the completeness of *CLc* in terms of properties of *Lc*. In this context *the conjunctive extension CLc of a concept language Lc is complete if the following two conditions hold:*

(1) Lc has a concept description that covers all the instances in the instance language Li; and

(2) for every instance $i \in Li$ there exists at least one concept description $c \in Lc$ that does not cover only this instance.

By theorem 4.8 below (1) implies that *CLc* has a conjunction that covers all the instances in *Li*. By theorem 4.9 below (2) implies that each strict subset of an instance language *Li* is covered by some conjunctions in *CLc*. Thus, if the conditions (1) and (2) hold, then by theorem 2.16 the conjunctive extension *CLc* is complete; i.e., every subset $I \subseteq Li$ is represented in *CLc* by at least one conjunction.

Theorem 4.8. *Consider a concept language Lc and its conjunctive extension CLc. Then:*

$$(\exists c \in Lc)(\forall i \in Li)M(c,i) \leftrightarrow (\exists C \in CLc)(\forall i \in Li)M_C(C,i).$$

Proof. ($\rightarrow$) The first part of the proof is a corollary of theorem 4.7.

($\leftarrow$) Consider a conjunction $C \in CLc$ such that $(\forall i \in Li)M_C(C,i)$. This is equivalent according to definition 4.4 to $(\forall i \in Li)(\forall c \in C)M(c,i)$. Thus, since $C \subseteq Lc$,

$$(\exists c \in Lc)(\forall i \in Li)M(c,i),$$

and the second part of the theorem is proven. □

Theorem 4.9. *Consider a concept language Lc and its conjunctive extension CLc. Then:*

$$(\forall i \in Li)(\exists c \in Lc)cons(c, \langle Li - \{i\}, \{i\}\rangle) \leftrightarrow (\forall I \subset Li)(\exists C \in CLc)cons_C(C, \langle I, Li - I\rangle).$$

Proof. ($\rightarrow$) Consider arbitrary $I \subset Li$. $(\forall i \in Li)(\exists c \in Lc)cons(c, \langle Li - \{i\}, \{i\}\rangle)$ implies the existence of a conjunction $C = \bigwedge_{i \in Li-I} c_i$ so that $c_i \in Lc$ and $cons(c_i, \langle Li - \{i\}, \{i\}\rangle)$. Thus, according to definition 2.22 we have that:

$$(\forall i \in I)(\forall c \in C)M(c,i) \tag{4.3}$$
$$(\forall i \in Li - I)(\exists c \in C)\neg M(c,i). \tag{4.4}$$

Since $Li \neq \emptyset$ and $C = \bigwedge_{i \in Li-I} c_i$, using constraint 2.11 and definition 4.3 we conclude that:

$$C \in CLc \tag{4.5}$$

According to definition 4.4 formula (4.3) implies $(\forall i \in I)M_C(C,i)$ and formula (4.4) implies $(\forall i \in Li - I)\neg M_C(C,i)$. Thus, using definition 4.5 we have that $cons_C(C, \langle I, Li - I\rangle)$. Combining the consistency predicate with (4.5) we have that:

$$(\exists C \in CLc)cons_C(C, \langle I, Li - I\rangle),$$

and the first part of the theorem is proven.

($\leftarrow$) Consider an arbitrary $i \in Li$. $(\forall I \subset Li)(\exists C \in CLc)cons_C(C, \langle I, Li - I\rangle)$ implies the existence of a conjunction $C \in CLc$ such that $cons_C(C, \langle Li - \{i\}, \{i\}\rangle)$. Thus, according to definition 4.5 we have that:

$$(\forall i \in Li - \{i\})M_C(C,i) \wedge \neg M_C(c,i).$$

Using definition 4.4 this conjunction is equivalent to:

$$(\forall i \in Li - \{i\})(\forall c \in C)M(c,i) \wedge (\exists c \in C)\neg M(c,i).$$

Thus,

$$(\exists c \in C)((\forall i \in Li - \{i\})M(c,i) \wedge \neg M(c,i)).$$

Since $C \subseteq Lc$ the last derivation implies:

$$(\exists c \in Lc)((\forall i \in Li - \{i\})M(c,i) \wedge \neg M(c,i)).$$

Using definition 2.22 this is equivalent to :

$$(\exists c \in Lc)cons(c, \langle Li - \{i\}, \{i\}\rangle).$$

Since the instance i has been arbitrarily chosen that we have proven that:

$$(\forall i \in Li)(\exists c \in Lc)cons(c, \langle Li - \{i\}, \{i\}\rangle),$$

and the second part of the theorem is proven. □

4.2 Conjunctive Version Spaces

Conjunctive version spaces are sets of all possible conjunctions in the conjunctive extensions of concept languages that are consistent with positive and negative training instances of target concepts. More formally:

Definition 4.10. (Conjunctive Version Space) *The conjunctive version space of a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ is defined as follows:*

$$CVS = \{C \in CLc \mid cons_C(C, \langle I^+, I^-\rangle)\},$$

where CLc is the conjunctive extension of the concept language Lc.

Conjunctive version spaces CVS are more complete than the ordinary version spaces VS. This property follows from the same property of the conjunctive extensions of concept languages (see theorem 4.7) and it is shown in corollary 4.11 given below. The corollary states that if an instance set $I \subseteq Li$ is covered by at least one concept description in a version space VS, then there exists a conjunction in the corresponding conjunctive version space CVS that covers the set I. In other words, if a concept is represented in a version space VS, then it is represented in the corresponding conjunctive version space CVS as well.

Corollary 4.11. *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with version space VS and conjunctive version space CVS. Then:*

$$(\forall I \subseteq Li)((\exists c \in VS)cons(c, \langle I, Li - I\rangle) \rightarrow (\exists C \in CVS)cons_C(C, \langle I, Li - I\rangle)).$$

Proof. The proof is analogous to that of theorem 4.7. □

Since the conjunctive version spaces are more complete than the ordinary version spaces, they are solutions of more concept learning tasks. In addition to that the conjunctive version spaces are not empty for all possible concept-learning tasks if and only if the conjunctive extensions of concept languages are complete (see corollary 4.12 below). Since the conjunctive extensions of concept languages are not necessarily complete, we conclude that *the conjunctive version spaces are a partial solution to the problem with incomplete concept languages* (the first part of our problem statement). This implies that in general the conjunctive version spaces can be used for classification of unseen instances using the rule of the unanimous vote of their conjunctions.

Corollary 4.12. *Consider an instance language Li, a concept language Lc, and a membership relation M. Then the conjunctive version spaces CVS for all possible tasks* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ *are not empty if and only if:*

$$(\forall I \subseteq Li)(\exists C \in CLc) cons_C(C, \langle I, Li - I \rangle),$$

where CLc is the conjunctive extension of the concept language Lc.

4.3 Conjunctive Version-Space Abstract Data Type

Conjunctive version spaces can be considered as an abstract data type. It can be defined analogously to the abstract data type of version spaces (see chapter 3). To distinguish the basic conjunctive version-space operations we denote them with the additional subscript *CVS* (e.g. $Update_{CVS}$).

We prove that the conjunctive version-space abstract data type can be expressed in terms of the version-space abstract data type. More precisely we show that:

- conjunctive version spaces can be represented by version spaces defined in ordinary concept languages (see subsection 4.3.1);
- most of the basic conjunctive version-space operations can be represented by algorithms which main operators are the basic version-space operations (see subsection 4.3.2).

4.3.1 Conjunctive Version Spaces in Concept Languages

Conjunctive version spaces can be represented in concept languages by a conjunctive set operator applied on two types of version spaces. The first type is version spaces with respect to negative instances. A version space with respect to a negative instance n is defined as a set of descriptions in a concept language Lc that are

consistent with the training set I^+ of positive instances and the instance n[2]. More formally:

Definition 4.13. (Version Space with respect to a Negative Instance) *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. Then a version space with respect to a negative instance $n \in I^-$ is defined as follows:*

$$\{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\}.$$

Notation 4.14. *A version space with respect to a negative instance $n \in I^-$ is denoted by $VS(n)$*[3].

Thus, $VS(n) = \{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\}$.

The second type of version spaces, the positive version space, is defined as a set of descriptions in a concept language Lc that are consistent with the training set I^+ of positive instances. More formally:

Definition 4.15. (Positive Version Space) *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. Then a positive version space is defined as follows:*

$$\{c \in Lc | (\forall p \in I^+) M(c, p)\}.$$

Notation 4.16. *A positive version space is denoted by $VS(\emptyset)$*[4].

Thus, $VS(\emptyset) = \{c \in Lc | (\forall p \in I^+) M(c, p)\}$.

From definitions 4.13 and 4.15 it is easy to see that version spaces $VS(n)$ with respect to negative instances are subsets of the positive version space $VS(\emptyset)$. This property is proven in corollary 4.17 given below.

Corollary 4.17. *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. Then:*

$$(\forall n \in I^-)(VS(n) \subseteq VS(\emptyset)).$$

[2] Version space with respect to positive instances can be defined by duality. To distinguish both types of these version spaces in the rest of the thesis every positive instance will be denoted by p and every negative instance will be denoted by n.

[3] Note that $VS(n)$ also depends on the set I^+ of positive instances, so $VS(I^+, n)$ would be a more complete notation, but where no misunderstanding is possible $VS(n)$ is used.

[4] Note that $VS(I^+, \emptyset)$ is a more complete notation, but where no misunderstanding is possible $VS(\emptyset)$ is used.

The other component needed to represent conjunctive version spaces in concept languages is a conjunctive set operator. We define it below.

Definition 4.18. (Conjunctive Set Operator *CS*) *Consider a concept language Lc, a set $S \subseteq Lc$ and an indexed family of sets $\{S_i\}_{i \in I}$ such that $(\forall i \in I)(S_i \subseteq Lc)$. Then the conjunctive set operator is defined as follows:*

$$CS(S, \{S_i\}_{i \in I}) = \{C \subseteq S | (\forall i \in I)(C \cap S_i \neq \emptyset)\}.$$

Given a set $S \subseteq Lc$ and an indexed family of sets $\{S_i\}_{i \in I}$ $(S_i \subseteq Lc)$ the conjunctive set operator generates all possible conjunctions C that are subsets of the set S and share at least one common conjunct with every set S_i.

Theorem 4.19 below states that a conjunctive version space can be represented by the conjunctive set operator applied on the corresponding positive version space and version spaces with respect to negative training instances.

Theorem 4.19. *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ with conjunctive version space CVS. Then:*

$$CVS = CS(VS(\emptyset), \{VS(n)\}_{n \in I^-}).$$

Proof.

$C \in CS(VS(\emptyset), \{VS(n)\}_{n \in I^-})$
iff (definition 4.18)
$(C \subseteq VS(\emptyset) \wedge (\forall n \in I^-)(C \cap VS(n) \neq \emptyset))$
iff (definitions 4.13 and 4.15)
$((\forall c \in C)(\forall p \in I^+)M(c,p) \wedge (\forall n \in I^-)(\exists c \in C)cons(c, \langle I^+, \{n\}\rangle))$
iff (definition 2.22)
$((\forall c \in C)(\forall p \in I^+)M(c,p) \wedge$
$(\forall n \in I^-)(\exists c \in C)((\forall p \in I^+)M(c,p) \wedge \neg M(c,n)))$
iff
$((\forall c \in C)(\forall p \in I^+)M(c,p) \wedge (\forall n \in I^-)(\exists c \in C)\neg M(c,n))$
iff (definition 4.4)
$((\forall p \in I^+)M_C(C,p) \wedge (\forall n \in I^-)\neg M_C(C,n))$
iff (definition 4.5)
$cons_C(C, \langle I^+, I^- \rangle)$
iff (definition 4.10)
$C \in CVS$

□

A simple corollary of theorem 4.19 is given below. It states that every conjunction $C \in CVS$ is a subset of the positive version space $VS(\emptyset)$. The corollary is used in theorem's proofs in the rest of the chapter.

Corollary 4.20. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with conjunctive version space CVS. Then:*

$$(\forall C \in CVS)(C \subseteq VS(\emptyset)).$$

Proof. The proof follows from definition 4.18 and theorem 4.19. □

By theorem 4.19 a conjunctive version space is uniquely represented in concept languages by a positive version space $VS(\emptyset)$ and version spaces $VS(n)$ with respect to negative instances. Thus, we consider the version spaces $VS(\emptyset)$ and $VS(n)$ as a representation of conjunctive version spaces that is formally introduced below.

Definition 4.21. (Conjunctive Version-Space Representation (CVSR)) *The conjunctive version space CVS of a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *is represented in the concept language Lc by an ordered pair* $\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$.

Note that the CVSR representation of a conjunctive version space contains a finite number of version spaces with respect to negative instances. This is due to constraint 2.21 that restricts the training sets to be finite.

4.3.2 Algorithms of the Basic Conjunctive Version-Space Operations

This subsection presents algorithms of the basic conjunctive version-space operations. They are considered for the case when the following two assumptions hold:

(1) conjunctive version spaces are given by their CVSR representations;

(2) every basic version-space operation has an algorithm.

Assumption (1) implies that conjunctive version spaces are represented by their positive version spaces and version spaces with respect to negative instances. Hence, in order to manipulate with them we need the basic version-space operations. According to assumption (2) each of these operations has an algorithm. Therefore, the basic conjunctive version-space operations are represented by algorithms which main operators are the basic version-space operations.

The algorithms of the basic conjunctive version-space operations are given with proofs and explanations. The order is different from that of the basic version-space operations in chapter 3. More precisely, the algorithm of the operation $Converged?_{CVS}$ is discussed after the algorithm of the operation $Classify_{CVS}$. The reason for this is that there exists a dependency between these algorithms in the context of conjunctive version spaces.

Algorithm *Initialise*$_{CVS}$
Input: *Lc*: a concept language.
Output: CVSR:$\langle VS(\emptyset), \emptyset \rangle$ of a conjunctive version space *CVS*.

$VS(\emptyset) = Initialise(Lc)$
$CVS = CS(VS(\emptyset), \emptyset)$
return $\langle VS(\emptyset), \emptyset \rangle$.

Figure 4.1: The Algorithm of the Operation *Initialise*$_{CVS}$.

4.3.2.1 Algorithm of the Operation *Initialise*$_{CVS}$

The operation *Initialise*$_{CVS}$ sets a conjunctive version space when not training instances are available. The algorithm of the operation is based on corollary 4.22. The corollary states that a conjunctive version space is represented by positive version space only if the training sets are empty.

Corollary 4.22. *Consider a task* $\langle Li, Lc, M, \langle \emptyset, \emptyset \rangle \rangle$ *and its conjunctive version space CVS. Then:*

$$CVS = CS(VS(\emptyset), \emptyset).$$

The algorithm of the operation *Initialise*$_{CVS}$ is presented in figure 4.1. Given a concept language *Lc* it forms the positive version space $VS(\emptyset)$. According to definition 3.3 the version space $VS(\emptyset)$ is generated by the basic version-space operation *Initialise*. The CVSR representation $\langle VS(\emptyset), \emptyset \rangle$ of the initial conjunctive version space *CVS* is returned as a result of the algorithm execution.

4.3.2.2 Algorithm of the Operation *Update*$_{CVS}$

The operation *Update*$_{CVS}$ revises the conjunctive version space of a target concept when a new instance is added to the training sets. The algorithm of the operation is based on theorems 4.23 and 4.24 given below. They determine procedures how to update the positive version space and the version spaces with respect to negative instances when a new training instance is given.

Theorem 4.23. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ *with conjunctive version space CVS , and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle \rangle$ *with conjunctive version space* CVS', *where* $I^{+\prime} = I^+ \cup \{\mathfrak{p}\}$. *If* $CVS = CS(VS(\emptyset), \{VS(n)\}_{n \in I^-})$ *then:*

$$CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^-})$$

where $VS'(\emptyset) = \{c \in VS(\emptyset) | M(c, \mathfrak{p})\}$
$VS'(n) = \{c \in VS(n) | M(c, \mathfrak{p})\}$ *for all* $n \in I^-$.

Proof. By theorem 4.19 $CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^-})$, where

$$VS'(\emptyset) = \{c \in Lc | (\forall p \in I^{+\prime}) M(c,p)\}$$
$$VS'(n) = \{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\} \text{ for all } n \in I^-.$$

Since $I^{+\prime} = I^+ \cup \{\mathfrak{p}\}$, by theorem 3.4:

$$VS'(\emptyset) = \{c \in VS(\emptyset) | M(c, \mathfrak{p})\}$$
$$VS'(n) = \{c \in VS(n) | M(c, \mathfrak{p})\} \text{ for all } n \in I^-.$$

□

Theorem 4.24. *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with conjunctive version space CVS, and a second task $\langle Li, Lc, M, \langle I^+, I^{-\prime}\rangle\rangle$ with conjunctive version space CVS', where $I^{-\prime} = I^- \cup \{\mathfrak{n}\}$. If $CVS = CS(VS(\emptyset), \{VS(n)\}_{n \in I^-})$ then:*

$$CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^{-\prime}})$$

where $VS'(\emptyset) = VS(\emptyset)$
$VS'(n) = VS(n)$ *for all* $n \in I^-$
$VS'(\mathfrak{n}) = \{c \in VS(\emptyset) | \neg M(c, \mathfrak{n})\}$.

Proof. By theorem 4.19 $CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^{-\prime}})$, where

$$VS'(\emptyset) = \{c \in Lc | (\forall p \in I^+) M(c,p)\} \tag{4.6}$$
$$VS'(n) = \{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\} \text{ for all } n \in I^- \tag{4.7}$$
$$VS'(\mathfrak{n}) = \{c \in Lc | cons(c, \langle I^+, \{\mathfrak{n}\}\rangle)\}. \tag{4.8}$$

Using definitions 4.13 and 4.15 (4.6) and (4.7) imply:

$$VS'(\emptyset) = VS(\emptyset)$$
$$VS'(n) = VS(n) \text{ for all } n \in I^-.$$

By theorem 3.5 (4.6) and (4.8) imply:

$$VS'(\mathfrak{n}) = \{c \in VS(\emptyset) | \neg M(c, \mathfrak{n})\}.$$

□

The algorithm of the operation $Update_{CVS}$ is presented in figure 4.2. It updates the CVSR representation $\langle VS(\emptyset), \{VS(n)\}_{n \in I^-}\rangle$ of a conjunctive version space CVS as follows.

Algorithm $Update_{CVS}$
Input: i: a new training instance.
CVSR:$\langle VS(\emptyset), \{VS(n)\}_{n\in I^-}\rangle$ of a conjunctive version space CVS.
Output:
CVSR:$\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^-}\rangle$ of a conjunctive version space CVS' s.t.
$CVS' = \{C \in CVS | M_C(C, \mathfrak{p})\}$ if i is a positive instance $\mathfrak{p}$.
CVSR:$\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^- \cup \{\mathfrak{n}\}}\rangle$ of a conjunctive version space CVS' s.t.
$CVS' = \{C \in CVS | \neg M_C(C, \mathfrak{n})\}$ if i is a negative instance $\mathfrak{n}$.

if instance i is a positive instance $\mathfrak{p}$ **then**
  $VS'(\emptyset) = Update(\mathfrak{p}, VS(\emptyset))$
  for $n \in I^-$ **do**
    $VS'(n) = Update(\mathfrak{p}, VS(n))$
return $\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^-}\rangle$
if instance i is a negative instance $\mathfrak{n}$ **then**
  $VS'(\emptyset) = VS(\emptyset)$
  for $n \in I^-$ **do**
    $VS'(n) = VS(n)$
  $VS'(\mathfrak{n}) = Update(\mathfrak{n}, VS(\emptyset))$
return $\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^- \cup \{\mathfrak{n}\}}\rangle$.

Figure 4.2: The Algorithm of the Operation $Update_{CVS}$.

- If a positive instance $\mathfrak{p}$ is given, the positive version space $VS(\emptyset)$ and the version spaces $VS(n)$ for all negative instances $n \in I^-$ are revised so that their descriptions inconsistent with the instance $\mathfrak{p}$ are removed. Thus, by theorem 4.23 the resulting version spaces $VS'(\emptyset)$ and $VS'(n)$ form a CVSR representation that characterises correctly the updated conjunctive version space CVS'.

- If a new negative training instance $\mathfrak{n}$ is given, the algorithm does not change the positive version space $VS(\emptyset)$ and the version spaces $VS(n)$ for all $n \in I^-$. It generates the version space $VS'(\mathfrak{n})$ with respect to the instance $\mathfrak{n}$. The version space $VS'(\mathfrak{n})$ is set equal to those descriptions in $VS(\emptyset)$ that do not cover the instance $\mathfrak{n}$. Thus, by theorem 4.24 the version spaces $VS(\emptyset)$ and $VS(n)$ as well as the resulting version space $VS'(\mathfrak{n})$ form a CVSR representation that characterises correctly the updated conjunctive version space CVS'.

Note that according to definition 3.6 each revision of version spaces $VS(\emptyset)$ and $VS(n)$ is realised by the basic version-space operation *Update*.

4.3.2.3 Algorithm of the Operation $Retraction_{CVS}$

The operation $Retraction_{CVS}$ revises the conjunctive version space of a target concept when an instance is removed from the training sets. The algorithm of the operation is based on theorems 4.25 and 4.26 given below. They determine procedures how to update the positive version space and the version spaces with respect to negative instances when a training instance is retracted.

Theorem 4.25. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with conjunctive version space* CVS*, and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with conjunctive version space* CVS'*, where* $I^{+\prime} = I^+ - \{\mathfrak{p}\}$ *and* $\mathfrak{p} \in I^+$*. If* $CVS = CS(VS(\emptyset), \{VS(n)\}_{n \in I^-})$ *then:*

$$CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^-})$$

where $VS'(\emptyset) = VS(\emptyset) \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, \{\mathfrak{p}\}\rangle)\}$
$VS'(n) = VS(n) \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, \{n\} \cup \{\mathfrak{p}\}\rangle)\}$ *for* $n \in I^-$.

Proof. By theorem 4.19 $CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^-})$, where

$$VS'(\emptyset) = \{c \in Lc | (\forall p \in I^{+\prime}) M(c, p)\}$$
$$VS'(n) = \{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\} \text{ for all } n \in I^-.$$

Since $I^{+\prime} = I^+ - \{\mathfrak{p}\}$ and $\mathfrak{p} \in I^+$, by theorem 3.7 :

$$VS'(\emptyset) = VS(\emptyset) \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, \{\mathfrak{p}\}\rangle)\}$$
$$VS'(n) = VS(n) \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, \{n\} \cup \{\mathfrak{p}\}\rangle)\} \text{ for all } n \in I^-.$$

□

Theorem 4.26. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with conjunctive version space* CVS*, and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle\rangle$ *with conjunctive version space* CVS'*, where* $I^{-\prime} = I^- - \{\mathfrak{n}\}$ *and* $\mathfrak{n} \in I^-$*. If* $CVS = CS(VS(\emptyset), \{VS(n)\}_{n \in I^-})$ *then:*

$$CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^{-\prime}})$$

where $VS'(\emptyset) = VS(\emptyset)$
$VS'(n) = VS(n)$ *for all* $n \in I^{-\prime}$.

Proof. By theorem 4.19 $CVS' = CS(VS'(\emptyset), \{VS'(n)\}_{n \in I^{-\prime}})$, where

$$VS'(\emptyset) = \{c \in Lc | (\forall p \in I^+) M(c, p)\}$$
$$VS'(n) = \{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\} \text{ for all } n \in I^{-\prime}.$$

Algorithm $Retraction_{CVS}$
Input: i: a training instance to be retracted.
CVSR:$\langle VS(\emptyset), \{VS(n)\}_{n\in I^-}\rangle$ of a conjunctive version space CVS.
Lc: a concept language.
I^+: a set of positive training instances.
I^-: a set of negative training instances.
Output:
CVSR:$\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^-}\rangle$ of a conjunctive version space CVS' s.t.
$CVS' = CVS \cup \{C \in CLc | cons_C(C, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$
if i is a positive instance $\mathfrak{p} \in I^+$.
CVSR:$\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^- - \{\mathfrak{n}\}}\rangle$ of a conjunctive version space CVS' s.t.
$CVS' = CVS \cup \{C \in CLc | cons_C(C, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\}$
if i is a negative instance $\mathfrak{n} \in I^-$.

if instance i is a positive instance $\mathfrak{p}$ **then**
  $VS'(\emptyset) = Retraction(\mathfrak{p}, VS(\emptyset), Lc, I^+, I^-)$
  for $n \in I^-$ **do**
    $VS'(n) = Retraction(\mathfrak{p}, VS(n), Lc, I^+, I^-)$
return $\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^-}\rangle$
if instance i is a negative instance $\mathfrak{n}$ **then**
  $VS'(\emptyset) = VS(\emptyset)$
  for $n \in I^- - \{\mathfrak{n}\}$ **do**
    $VS'(n) = VS(n)$
return $\langle VS'(\emptyset), \{VS'(n)\}_{n\in I^- - \{\mathfrak{n}\}}\rangle$.

Figure 4.3: The Algorithm of the Operation $Retraction_{CVS}$.

Thus, according to definitions 4.13 and 4.15:

$$VS'(\emptyset) = VS(\emptyset)$$
$$VS'(n) = VS(n) \text{ for all } n \in I^{-\prime}.$$

□

The algorithm of the operation $Retraction_{CVS}$ is given in figure 4.3. It revises the CVSR representation $\langle VS(\emptyset), \{VS(n)\}_{n\in I^-}\rangle$ of a conjunctive version space CVS as follows.

- If a positive training instance $\mathfrak{p}$ is given, the positive version space $VS(\emptyset)$ and the version spaces $VS(n)$ for all negative instances $n \in I^-$ are modified by

Algorithm $Collapsed?_{CVS}$
Input: CVSR:$\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$ of a conjunctive version space CVS.
Output: true if $CVS = \emptyset$.
false if $CVS \neq \emptyset$.

```
if Collapsed?(VS(∅)) then
    return true
for n ∈ I⁻ do
    if Collapsed?(VS(n)) then
        return true
return false.
```

Figure 4.4: The Algorithm of the Operation $Collapsed?_{CVS}$.

the basic version-space operation *Retraction* specified in definition 3.9. Thus, by theorem 4.25 the revised version spaces $VS'(\emptyset)$ and $VS'(n)$ form a CVSR representation that characterises correctly the resulting conjunctive version space CVS'.

- If a negative training instance $\mathfrak{n}$ is retracted, the algorithm does not change the positive version space $VS(\emptyset)$ and the version spaces $VS(n)$ for all $n \in I^- - \{\mathfrak{n}\}$. It removes the version space $VS(\mathfrak{n})$ of the instance $\mathfrak{n}$. Thus, by theorem 4.26 the version spaces $VS(\emptyset)$ and $VS(n)$ form a CVSR representation that characterises correctly the resulting conjunctive version space CVS'.

4.3.2.4 Algorithm of the Operation $Collapsed?_{CVS}$

The operation $Collapsed?_{CVS}$ determines whether a conjunctive version space is empty. The algorithm of the operation is based on corollary 4.27. The corollary states that the conjunctive version space is empty if and only if the positive version space or at least one version space with respect to a negative instance is empty.

Corollary 4.27. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ *and its conjunctive version space* $CVS = CS(VS(\emptyset), \{VS(n)\}_{n \in I^-})$. *Then:*

$$(CVS = \emptyset) \leftrightarrow (VS(\emptyset) = \emptyset \vee (\exists n \in I^-)(VS(n) = \emptyset)).$$

Proof. The proof follows from definition 4.18 and theorem 4.19. □

The algorithm of the operation $Collapsed?_{CVS}$ is given in figure 4.4. Its input is the CVSR representation $\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$ of a conjunctive version space CVS.

To test CVS for a collapse the algorithm tests the corresponding version spaces $VS(\emptyset)$ and $VS(n)$ for a collapse. If at least one of the version spaces $VS(\emptyset)$ and $VS(n)$ is empty, then by corollary 4.27 the conjunctive version space CVS is empty and the algorithm returns true. Otherwise, CVS is not empty and the algorithm returns false. According to definition 3.10 the test of the version spaces $VS(\emptyset)$ and $VS(n)$ for a collapse is realised by the basic version-space operation *Collapsed?*.

4.3.2.5 Algorithm of the Operation $Classify_{CVS}$

The operation $Classify_{CVS}$ determines the classification of an instance using the unanimous vote of conjunctions in a conjunctive version space. The algorithm of the operation is based on theorems 4.28 and 4.29. The theorem 4.28 states that an instance is covered by all the conjunctions of a conjunctive version space if and only if the instance is covered by all the descriptions of version space $VS(\emptyset)$. The theorem 4.29 states that an instance is not covered by any conjunction of a conjunctive version space if and only if the instance is not covered by any description of at least one version space $VS(n)$.

Theorem 4.28. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated conjunctive version space CVS. Then:*

$$(\forall i \in Li)((\forall C \in CVS) M_C(C, i) \leftrightarrow (\forall c \in VS(\emptyset)) M(c, i)).$$

Proof. The statement of the theorem is equivalent to:

$$(\forall i \in Li)((\exists C \in CVS) \neg M_C(C, i) \leftrightarrow (\exists c \in VS(\emptyset)) \neg M(c, i)).$$

Thus, we prove the theorem by proving the last equivalence.

($\rightarrow$) Choose an arbitrary instance $i \in Li$. Consider a particular conjunction $C \in CVS$ such that $\neg M_C(C, i)$. This implies according to definition 4.4 that $(\exists c \in C) \neg M(c, i)$. But, by corollary 4.20 $C \subseteq VS(\emptyset)$. Thus, we conclude that $(\exists c \in VS(\emptyset)) \neg M(c, i)$, and the first part of the theorem is proven.

($\leftarrow$) Choose an arbitrary instance $i \in Li$. Consider a particular description $c \in VS(\emptyset)$ such that $\neg M(c, i)$. According to definition 4.18 and theorem 4.19 $c \in VS(\emptyset)$ implies that there exists a conjunction $C \in CVS$ such that $c \in C$. Using definition 4.4 $c \in C$ and $\neg M(c, i)$ imply $\neg M_C(C, i)$. Thus, $(\exists C \in CVS) \neg M_C(C, i)$, and the second part of the theorem is proven.

□

Theorem 4.29. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated conjunctive version space CVS. Then:*

$$(\forall i \in Li)((\forall C \in CVS) \neg M_C(C, i) \leftrightarrow (\exists n \in I^-)(\forall c \in VS(n)) \neg M(c, i)).$$

Proof. The statement of the theorem is equivalent to:

$$(\forall i \in Li)((\exists C \in CVS)M_C(C,i) \leftrightarrow (\forall n \in I^-)(\exists c \in VS(n))M(c,i)).$$

Thus, we prove the theorem by proving the last equivalence.

($\rightarrow$) Choose an arbitrary instance $i \in Li$. Consider a particular conjunction $C \in CVS$ such that $M_C(C,i)$. Using definition 4.4 $M_C(C,i)$ implies $(\forall c \in C)M(c,i)$. Since $C \in CVS$ it follows according to theorem 4.19 and definition 4.18 that $(\forall n \in I^-)(VS(n) \cap C \neq \emptyset)$. Thus, from the last two derivations we have that:

$$(\forall n \in I^-)(\exists c \in C \cap VS(n))M(c,i).$$

This implies:

$$(\forall n \in I^-)(\exists c \in VS(n))M(c,i),$$

and the first part of the theorem is proven.

($\leftarrow$) Choose an arbitrary instance $i \in Li$ such that $(\forall n \in I^-)(\exists c \in VS(n))M(c,i)$. Let us construct a conjunction C that contains at least one conjunct $c \in VS(n)$ for each $n \in I^-$ such that $M(c,i)$. By corollary 4.17 $(\forall c \in C)(c \in VS(\emptyset))$. Thus, by theorem 4.19 it follows that $C \in CVS$ and by the construction of the conjunction C it follows that $(\forall c \in C)M(c,i)$. This implies $M_C(C,i)$ according to definition 4.4. Thus, $(\exists C \in CVS)M_C(C,i)$, and the second part of the theorem is proven. □

The algorithm of the operation $Classify_{CVS}$ is given in figure 4.5. Its input is an instance i to be classified, and the CVSR representation $\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$ of a conjunctive version space CVS. The instance classification starts with a test that determines whether the conjunctive version space CVS is empty. If so, then according to definition 3.14 the algorithm returns "?". Otherwise, the algorithm determines whether the instance is covered by all the descriptions of the positive version space $VS(\emptyset)$. If so, by theorem 4.28 the instance is covered by all the conjunctions in the conjunctive version space, and according to definition 3.14 the algorithm returns "+". Otherwise, the algorithm visits the version spaces $VS(n)$ with respect to negative instances. If all the descriptions of at least one version space $VS(n)$ do not cover the instance, then by theorem 4.29 the instance is not covered by all the conjunctions in the conjunctive version space. Hence, according to definition 3.14 the algorithm returns "−". If neither a positive nor a negative classification of the instance is determined, the algorithm returns "?".

Note that according to definition 3.14 the algorithm checks whether all the descriptions of the version spaces $VS(\emptyset)$ and $VS(n)$ do (not) cover the instance by the basic version-space operation *Classify*.

Algorithm *Classify*$_{CVS}$

Input: i: an instance to be classified.
CVSR:$\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$ of a conjunctive version space CVS.
Output: "+" if $(CVS \neq \emptyset) \wedge (\forall C \in CVS) M_C(C, i)$.
"−" if $(CVS \neq \emptyset) \wedge (\forall C \in CVS) \neg M_C(C, i)$.
"?" otherwise.

if *Collapsed?*$_{CVS}(\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle)$ **then**
  return "?"
if *Classify*$(i, VS(\emptyset))$= "+" **then**
  return "+"
for $n \in I^-$ **do**
  if *Classify*$(i, VS(n))$= "−" **then**
    return "−"
return "?"

Figure 4.5: The Algorithm of the Operation *Classify*$_{CVS}$.

4.3.2.6 Algorithm of the Operation *Converged?*$_{CVS}$

The operation *Converged?*$_{CVS}$ determines when a conjunctive version space consists of equivalent conjunctions. Conjunctions are equivalent if and only if an equivalence relation holds. The relation is specified in definition 4.30 given below.

Definition 4.30. (Equivalence Relation on *CLc* ($\sim_C$)) *Consider an instance language Li, the conjunctive extension CLc of a concept language Lc and a membership relation M. Then:*

$$(\forall C_1, C_2 \in CLc)((C_1 \sim_C C_2) \leftrightarrow (\forall i \in Li)(M_C(C_1, i) \leftrightarrow M_C(C_2, i))).$$

The definition states that conjunctions $C_1, C_2 \in CLc$ are equivalent $(C_1 \sim_C C_2)$ if and only if the equivalency $M_C(C_1, i) \leftrightarrow M_C(C_2, i)$ holds for all instances $i \in Li$. In other words, the conjunctions are equivalent if and only if the extensional representations of the concepts, that they stand for, are equal.

A conjunctive version space consists of equivalent conjunctions if and only if there exists a set $I \subseteq Li$ such that all the conjunctions are consistent with the set I considered as positive and the set $Li - I$ considered as negative. This property is proven in theorem 4.31 given below.

Theorem 4.31. *Consider a conjunctive version space CVS. Then:*

$$(\forall C_1, C_2 \in CVS)(C_1 \sim_C C_2) \leftrightarrow (\exists I \subseteq Li)(\forall C \in CVS) cons_C(C, \langle I, Li - I \rangle).$$

Proof. The proof follows from definition 4.30. □

Corollary 4.32, given below, of theorems 4.31, 4.28 and 4.29 states that a conjunctive version space consists of equivalent conjunctions if and only if there exists a set $I \subseteq Li$ such that (1) the instances of I are covered by all the descriptions of the positive version space $VS(\emptyset)$, and (2) each of the instances in the set $Li - I$ is rejected by all the descriptions of at least one version space $VS(n)$ with respect to a negative instance n.

Corollary 4.32. *Consider a conjunctive version space CVS of a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$. *Then:*

$$\begin{aligned}(\forall C_1, C_2 \in CVS)(C_1 \sim_C C_2) \leftrightarrow \\ (\exists I \subseteq Li)((\forall i \in I)(\forall c \in VS(\emptyset))M(c,i) \wedge \\ (\forall i \in Li - I)(\exists n \in I^-)(\forall c \in VS(n))\neg M(c,i)).\end{aligned}$$

Corollary 4.32 can be used for building an algorithm of the operation $Converged?_{CVS}$ when the instance language Li is finite. Given a non-empty conjunctive version space CVS the algorithm operates as follows. It visits sequentially the instances $i \in Li$. If every instance i is either covered by all the descriptions of the version space $VS(\emptyset)$ or rejected by all the descriptions of at least one version space $VS(n)$, then by corollary 4.32 this is an indication that CVS consists of equivalent conjunctions. Hence, the algorithm returns true. Otherwise, it returns false.

The restriction on the instance language Li to be finite is too strong. Thus, when Li is infinite there is no algorithm of the operation $Converged?_{CVS}$ that is based on the basic version-space operations. This is a corollary of a fact that determining equivalence is an undecidable problem in general (Sablon, 1995). That is why for the sake of simplification of the presentation *we exclude the operation* $Converged?_{CVS}$ *from the set of the operations of the abstract data type of conjunctive version spaces.*

4.3.2.7 Algorithm of the Operation $Member?_{CVS}$

The operation $Member?_{CVS}$ determines whether a conjunction belongs to a conjunction version space. The algorithm of the operation is based on corollary 4.33 below. The corollary states that a conjunction C belongs to a conjunctive version space CVS if and only if every version space $VS(n)$ has an element in C and C is a subset of the positive version space $VS(\emptyset)$.

Algorithm $Member?_{CVS}$

Input: C: a finite conjunction.
CVSR:$\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$ of a conjunctive version space CVS.
Output: true if $(C \in CVS)$.
false if $\neg(C \in CVS)$.

```
for each n ∈ I⁻ do
    found = false
    for each c ∈ C do
        if Member?(c, VS(n)) then
            found = true
    if ¬found then
        return false
for each c ∈ C do
    if ¬Member?(c, VS(∅)) then
        return false
return true.
```

Figure 4.6: The Algorithm of the Operation $Member?_{CVS}$.

Corollary 4.33. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ *with conjunctive version space CVS. If CLc is the conjunctive extension of Lc then:*

$$(\forall C \in CLc)((C \in CVS) \leftrightarrow ((\forall n \in I^-)(\exists c \in C)(c \in VS(n)) \wedge (C \subseteq VS(\emptyset)))).$$

Proof. The proof follows from definition 4.18 and theorem 4.19. □

The algorithm of the operation $Member?_{CVS}$ is given in figure 4.6. Its input is a finite conjunction C, and the CVSR representation $\langle VS(\emptyset), \{VS(n)\}_{n \in I^-} \rangle$ of a conjunctive version space CVS. To determine the membership of C to CVS the algorithm checks whether:

(1) each version space $VS(n)$ contains at least one conjunct $c \in C$; and

(2) each conjunct $c \in C$ is in the version space $VS(\emptyset)$.

If the conditions (1) and (2) do hold then by corollary 4.33 C is a member of CVS and the algorithm returns true. Otherwise, C is not a member of CVS and the algorithm returns false.

According to definition 3.18 in order to determine the membership of each conjunct $c \in C$ to either the version spaces $VS(n)$ or a version space $VS(\emptyset)$ the algorithm employs the basic version-space operation *Member?*. Note that the algorithm does not use the basic version-space operation *Subset?* to determine that $C \subseteq VS(\emptyset)$. This is due to the fact that the conjunction C is not necessarily a version space.

4.3.2.8 Algorithm of the Operation $Intersection_{CVS}$

The operation $Intersection_{CVS}$ computes a conjunctive version space CVS_{12} that is the intersection of two other conjunctive version spaces CVS_1 and CVS_2. The algorithm of the operation is based on theorem 4.34. The theorem determines how to compute the version spaces $VS_{12}(\emptyset)$ and $VS_{12}(n)$ from version spaces $VS_1(\emptyset)$, $VS_1(n)$, $VS_2(\emptyset)$, and $VS_2(n)$.

Theorem 4.34. *Consider a task $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ with conjunctive version space CVS_1; a second task $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ with conjunctive version space CVS_2; and a third task $\langle Li, Lc, M, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^- \rangle\rangle$ with conjunctive version space CVS_{12}. If $CVS_1 = CS(VS_1(\emptyset), \{VS_1(n)\}_{n \in I_1^-})$ and $CVS_2 = CS(VS_2(\emptyset), \{VS_2(n)\}_{n \in I_2^-})$ then:*

$$CVS_{12} = CS(VS_{12}(\emptyset), \{VS_{12}(n)\}_{n \in I_1^- \cup I_2^-})$$

where $VS_{12}(\emptyset) = VS_1(\emptyset) \cap VS_2(\emptyset)$

$$VS_{12}(n) = \begin{cases} VS_1(n) \cap VS_2(\emptyset) & \text{if } n \in I_1^- \\ VS_2(n) \cap VS_1(\emptyset) & \text{if } n \in I_2^-. \end{cases}$$

Proof. By theorem 4.19 $CVS_{12} = CS(VS_{12}(\emptyset), \{VS_{12}(n)\}_{n \in I_1^- \cup I_2^-})$. Hence, we have to determine the version spaces $VS_{12}(\emptyset)$ and $VS_{12}(n)$.

1). We determine the version space $VS_{12}(\emptyset)$. According to definition 4.15:

$$\begin{aligned} VS_{12}(\emptyset) &= \{c \in Lc | (\forall p \in I_1^+ \cup I_2^+) M(c,p)\} \\ VS_1(\emptyset) &= \{c \in Lc | (\forall p \in I_1^+) M(c,p)\} \\ VS_2(\emptyset) &= \{c \in Lc | (\forall p \in I_2^+) M(c,p)\}. \end{aligned}$$

Thus, by theorem 3.20 we have:

$$VS_{12}(\emptyset) = VS_1(\emptyset) \cap VS_2(\emptyset).$$

2). We determine the version spaces $VS_{12}(n)$ for $n \in I_1^- \cup I_2^-$. According to definition 4.13:

$$VS_{12}(n) = \{c \in Lc | cons(c, \langle I_1^+ \cup I_2^+, \{n\}\rangle)\}.$$

2.1). If $n \in I_1^-$ then by theorem 3.20 we have that:

$$VS_{12}(n) = VS_1(n) \cap VS_2(\emptyset)$$

where $VS_1(n) = \{c \in Lc | cons(c, \langle I_1^+, \{n\}\rangle)\}$.

2.2). If $n \in I_2^-$ then by theorem 3.20 we have that:

$$VS_{12}(n) = VS_2(n) \cap VS_1(\emptyset)$$

where $VS_2(n) = \{c \in Lc | cons(c, \langle I_2^+, \{n\}\rangle)\}$. □

In principle theorem 4.34 can be used as a basis of an algorithm of the operation $Intersection_{CVS}$. The main problem of this algorithm is computing twice the version spaces $VS_{12}(n)$ for negative instances $n \in I_1^- \cap I_2^-$. We resolve this problem in a simple corollary of the theorem that is given below.

Corollary 4.35. *Consider a task $\langle Li, Lc, M, \langle I_1^+, I_1^-\rangle\rangle$ with conjunctive version space CVS_1; a second task $\langle Li, Lc, M, \langle I_2^+, I_2^-\rangle\rangle$ with conjunctive version space CVS_2; and a third task $\langle Li, Lc, M, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^-\rangle\rangle$ with conjunctive version space CVS_{12}. If $CVS_1 = CS(VS_1(\emptyset), \{VS_1(n)\}_{n \in I_1^-})$ and $CVS_2 = CS(VS_2(\emptyset), \{VS_2(n)\}_{n \in I_2^-})$ then:*

$$CVS_{12} = CS(VS_{12}(\emptyset), \{VS_{12}(n)\}_{n \in I_1^- \cup I_2^-})$$

where
$$VS_{12}(\emptyset) = VS_1(\emptyset) \cap VS_2(\emptyset)$$
$$VS_{12}(n) = \begin{cases} VS_1(n) \cap VS_2(\emptyset) & \text{if } n \in I_1^- \\ VS_2(n) \cap VS_1(\emptyset) & \text{otherwise.} \end{cases}$$

The corollary determines the algorithm of the operation $Intersection_{CVS}$. The algorithm is given in figure 4.7. Its input is the CVSR representations $\langle VS_1(\emptyset), \{VS_1(n)\}_{n \in I_1^-}\rangle$ and $\langle VS_2(\emptyset), \{VS_2(n)\}_{n \in I_2^-}\rangle$ of conjunctive version spaces CVS_1 and CVS_2. To intersect CVS_1 and CVS_2 the algorithm executes the following three steps. In the first step the algorithm computes the positive version space $VS_{12}(\emptyset)$. It is set equal to the intersection of the positive version spaces $VS_1(\emptyset)$ and $VS_2(\emptyset)$. In the second step the algorithm computes the version spaces $VS_{12}(n)$ for negative instances $n \in I_1^-$. Each of them is set equal to the intersection of the corresponding version space $VS_1(n)$ and the positive version space $VS_2(\emptyset)$. In the third step the algorithm computes the version spaces $VS_{12}(n)$ for negative instances $n \in I_2^- - I_1^-$. Each of them is set equal to the intersection of the corresponding version space $VS_2(n)$ and the positive version space $VS_1(\emptyset)$. Thus, by theorem 4.34 the resulting CVSR representation $\langle VS_{12}(\emptyset), \{VS_{12}(n)\}_{n \in I_2^- \cup I_2^-}\rangle$ characterises correctly the intersected conjunctive version space CVS_{12}.

Note that according to definition 3.19 all intersections of the version spaces $VS_1(\emptyset)$, $VS_1(n)$, $VS_2(\emptyset)$, and $VS_2(n)$ are realised by the basic version-space operation *Intersection*.

Algorithm $Intersection_{CVS}$

Input: CVSR:$\langle VS_1(\emptyset), \{VS_1(n)\}_{n\in I_1^-}\rangle$ of conjunctive version space CVS_1.
CVSR:$\langle VS_2(\emptyset), \{VS_2(n)\}_{n\in I_2^-}\rangle$ of conjunctive version space CVS_2.
Output: CVSR:$\langle VS_{12}(\emptyset), \{VS_{12}(n)\}_{n\in I_2^- \cup I_2^-}\rangle$ of conjunctive version space $CVS_{12} = CVS_1 \cap CVS_2$.

$VS_{12}(\emptyset) = Intersection(VS_1(\emptyset), VS_2(\emptyset))$
for $n \in I_1^-$ **do**
  $VS_{12}(n) = Intersection(VS_1(n), VS_2(\emptyset))$
for $n \in I_2^- - I_1^-$ **do**
  $VS_{12}(n) = Intersection(VS_2(n), VS_1(\emptyset))$
return $\langle VS_{12}(\emptyset), \{VS_{12}(n)\}_{n\in I_2^- \cup I_2^-}\rangle$.

Figure 4.7: The Algorithm of the Operation $Intersection_{CVS}$.

4.3.2.9 Algorithm of the Operation $Subset?_{CVS}$

The operation $Subset?_{CVS}$ determines whether a conjunctive version space CVS_1 is a subset of another conjunctive version space CVS_2. The algorithm of the operation is based on theorem 4.36. The theorem states that CVS_1 is a subset of CVS_2 if and only if for each version space $VS_1(n)$ there exists version space $VS_2(n)$ such that $VS_1(n) \subseteq VS_2(n)$ and the positive version space $VS_1(\emptyset)$ is a subset of the positive version space $VS_2(\emptyset)$.

Theorem 4.36. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^-\rangle\rangle$ *with conjunctive version space* CVS_1*, and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^-\rangle\rangle$ *with conjunctive version space* CVS_2*. If* $CVS_1 = CS(VS_1(\emptyset), \{VS_1(n)\}_{n\in I_1^-})$ *and* $CVS_2 = CS(VS_2(\emptyset), \{VS_2(n)\}_{n\in I_2^-})$ *then:*

$$(CVS_1 \subseteq CVS_2) \leftrightarrow ((\forall n_2 \in I_2^-)(\exists n_1 \in I_1^-)(VS_2(n_2) \supseteq VS_1(n_1)) \wedge (VS_2(\emptyset) \supseteq VS_1(\emptyset))).$$

Proof. ($\rightarrow$) $CVS_1 \subseteq CVS_2$ implies $CVS_1 = CVS_1 \cap CVS_2$. Thus, by theorem 4.34 $CVS_1 = CS(VS_1(\emptyset), \{VS_1(n)\}_{n\in I_1^-})$ such that:

$$VS_1(\emptyset) = VS_1(\emptyset) \cap VS_2(\emptyset) \tag{4.9}$$

$$VS_1(n) = \begin{cases} VS_1(n) \cap VS_2(\emptyset) & \text{if } n \in I_1^- \\ VS_2(n) \cap VS_1(\emptyset) & \text{if } n \in I_2^-. \end{cases} \tag{4.10}$$

Formula (4.9) implies:

$$VS_1(\emptyset) \subseteq VS_2(\emptyset) \tag{4.11}$$

Formula (4.10) implies:

$$(\forall n_2 \in I_2^-)(\exists n_1 \in I_1^-)(VS_2(n_2) \cap VS_1(\emptyset) = VS_1(n_1)).$$

Thus,

$$(\forall n_2 \in I_2^-)(\exists n_1 \in I_1^-)(VS_2(n_2) \supseteq VS_1(n_1)) \tag{4.12}$$

From (4.11) and (4.12) the first part of the theorem is proven.

($\leftarrow$) Consider the first part of the right-hand side:

$$\begin{aligned}
&(\forall n_2 \in I_2^-)(\exists n_1 \in I_1^-)(VS_2(n_2) \supseteq VS_1(n_1)) \\
&\text{ implies (by theorem 3.22):} \\
&(\forall n_2 \in I_2^-)(\exists n_1 \in I_1^-)(\forall c \in VS_1(n_1))\neg M(c, n_2) \\
&\text{ which is equivalent (by theorem 4.29) to:} \\
&(\forall n_2 \in I_2^-)(\forall C \in CVS_1)\neg M_C(C, n_2)
\end{aligned} \tag{4.13}$$

Consider the second part of the right-hand side:

$$\begin{aligned}
&VS_2(\emptyset) \supseteq VS_1(\emptyset) \\
&\text{ is equivalent (by theorem 3.22) to:} \\
&(\forall p_2 \in I_2^+)(\forall c \in VS_1(\emptyset))M(c, p_2) \\
&\text{ which is equivalent (by theorem 4.28) to:} \\
&(\forall p_2 \in I_2^+)(\forall C \in CVS_1)M_C(C, p_2)
\end{aligned} \tag{4.14}$$

By theorem 3.22 formulas (4.13) and (4.14) imply:

$$CVS_1 \subseteq CVS_2.$$

Thus, the second part of the theorem is proven. □

The algorithm of the operation $Subset?_{CVS}$ is given in figure 4.8. Its input is the CVSR representations $\langle VS_1(\emptyset), \{VS_1(n)\}_{n \in I_1^-} \rangle$ and $\langle VS_2(\emptyset), \{VS_2(n)\}_{n \in I_2^-} \rangle$ of conjunctive version spaces CVS_1 and CVS_2. To test whether CVS_1 is a subset of CVS_2 the algorithm executes two main steps. In the first step it checks whether each version space $VS_1(n_1)$ is a subset of at least one version space $VS_2(n_2)$. If

Algorithm *Subset?*$_{CVS}$

Input: CVSR:$\langle VS_1(\emptyset), \{VS_1(n)\}_{n \in I_1^-} \rangle$ of conjunctive version space CVS_1.
CVSR:$\langle VS_2(\emptyset), \{VS_2(n)\}_{n \in I_2^-} \rangle$ of conjunctive version space CVS_2.
Output: true if $(CVS_1 \subseteq CVS_2)$.
false if $\neg(CVS_1 \subseteq CVS_2)$.

for each $n_1 \in I_1^-$ **do**
  found = false
  for each $n_2 \in I_2^-$ **do**
    if *Subset?*$(VS_1(n_1), VS_2(n_2))$ **then**
      found = true
  if $\neg$*found* **then**
    return false
if $\neg$*Subset?*$(VS_1(\emptyset), VS_2(\emptyset))$ **then**
  return false
return true.

Figure 4.8: The Algorithm of the Operation *Subset?*$_{CVS}$.

not, by theorem 4.36 CVS_1 is not a subset of CVS_2 and the algorithm returns false. Otherwise, the algorithm executes the second step: it checks whether the positive version space $VS_1(\emptyset)$ is a subset of the positive version space $VS_2(\emptyset)$. If so, by theorem 4.36 CVS_1 is a subset of CVS_2 and the algorithm returns true. Otherwise, it returns false; i.e., CVS_1 is not a subset of CVS_2.

Note that according to definition 3.21 all subset tests are realised by the basic version-space operation *Subset?*.

4.3.2.10 Algorithm of the Operation *Equal?*$_{CVS}$

The operation *Equal?*$_{CVS}$ determines whether a conjunctive version space CVS_1 is equal to another conjunctive version space CVS_2. The algorithm of the operation is based on corollary 4.37. The corollary states that CVS_1 is a subset of CVS_2 if and only if for each version space $VS_1(n_1)$ $(n_1 \in I_1^-)$ there exists a version space $VS_2(n_2)$ $(n_2 \in I_2^-)$ such that $VS_1(n_1) \subseteq VS_2(n_2)$; for each version space $VS_2(n_2)$ there exists a version space $VS_1(n_1)$ such that $VS_2(n_2) \subseteq VS_1(n_1)$; and the positive version space $VS_1(\emptyset)$ is equal to the positive version space $VS_2(\emptyset)$.

Corollary 4.37. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle \rangle$ *with conjunctive version space* CVS_1*, and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle \rangle$ *with con-*

Algorithm $Equal?_{CVS}$

Input: CVSR:$\langle VS_1(\emptyset), \{VS_1(n)\}_{n\in I_1^-}\rangle$ of conjunctive version space CVS_1.
CVSR:$\langle VS_2(\emptyset), \{VS_2(n)\}_{n\in I_2^-}\rangle$ of conjunctive version space CVS_2.
Output: true if $(CVS_1 = CVS_2)$.
false if $\neg(CVS_1 = CVS_2)$.

```
for each n1 ∈ I1⁻ do
    found = false
    for each n2 ∈ I2⁻ do
        if Subset?(VS1(n1), VS2(n2)) then
            found = true
    if ¬found then
        return false
for each n2 ∈ I2⁻ do
    found = false
    for each n1 ∈ I1⁻ do
        if Subset?(VS2(n2), VS1(n1)) then
            found = true
    if ¬found then
        return false
if ¬Equal?(VS1(∅), VS2(∅)) then
    return false
return true.
```

Figure 4.9: The Algorithm of the Operation $Equal?_{CVS}$.

junctive version space CVS_2. *If* $CVS_1 = CS(VS_1(\emptyset), \{VS_1(n)\}_{n\in I_1^-})$ *and* $CVS_2 = CS(VS_2(\emptyset), \{VS_2(n)\}_{n\in I_2^-})$ *then:*

$$\begin{aligned}(CVS_1 = CVS_2) \leftrightarrow ((\forall n_2 \in I_2^-)(\exists n_1 \in I_1^-)(VS_2(n_2) \supseteq VS_1(n_1)) \quad \wedge \\ (\forall n_1 \in I_1^-)(\exists n_2 \in I_2^-)(VS_1(n_1) \supseteq VS_2(n_2)) \quad \wedge \\ (VS_2(\emptyset) = VS_1(\emptyset))).\end{aligned}$$

Proof. The proof follows from theorem 4.36 □

The algorithm of the operation $Equal?_{CVS}$ is given in figure 4.9. Its input is the CVSR representations $\langle VS_1(\emptyset), \{VS_1(n)\}_{n\in I_1^-}\rangle$ and $\langle VS_2(\emptyset), \{VS_2(n)\}_{n\in I_2^-}\rangle$ of

conjunctive version spaces CVS_1 and CVS_2. To test whether CVS_1 and CVS_2 are equal the algorithm executes three main steps. In the first step it checks whether each version space $VS_1(n_1)$ is a subset of at least one version space $VS_2(n_2)$. If not, by corollary 4.37 CVS_1 is not equal to CVS_2 and the algorithm returns false. Otherwise, the algorithm executes the second step. In this step it checks whether each version space $VS_2(n_2)$ is a subset of at least one version space $VS_1(n_1)$. If not, by theorem 4.37 CVS_1 is not equal to CVS_2 and the algorithm returns false. Otherwise, the algorithm executes the third step where it checks whether the positive version space $VS_1(\emptyset)$ is equal the positive version space $VS_2(\emptyset)$. If so, by theorem 4.37 CVS_1 is equal to CVS_2 and the algorithm returns true. Otherwise it returns false; i.e., CVS_1 is not equal to CVS_2.

Note that according to definitions 3.21 and 3.23 all subset and equality operations are realised by the basic version-space operations *Subset?* and *Equal?*, respectively.

4.4 Disjunctive Case

Disjunctive extensions of concept languages, disjunctive version spaces and their abstract data type can be derived by duality from the previous sections. In order to avoid repetitions we leave out their descriptions.

4.5 Chapter Conclusion

The first contribution of this chapter is that it provides a partial solution to the problem of incomplete concept languages. The core idea has been to extend concept languages with logical conjunction[5]. It has been proven that the conjunctive extensions of concept languages are more complete than the languages themselves. This has motivated introducing the conjunctive version spaces. They have been defined as version spaces in the conjunctive extensions of concept languages. The completeness properties of the conjunctive version spaces have been analysed. We have proven that the conjunctive version spaces are more complete than the (ordinary) version spaces. Hence, they are solutions of more concept-learning tasks. In addition to that we have proven that the conjunctive version spaces are nonempty for all possible concept-learning tasks if and only if the conjunctive extensions of concept languages are complete. Since the conjunctive extensions of concept languages are not necessarily complete, it has been determined that *the conjunctive*

[5] Note that the idea to extend concept languages with logical conjunction in order to improve their completeness is new in the context of version space research. The previous works of Mitchell (1978), Murray (1987), Smirnov (1992), Sablon (1995), Smirnov and Neves (1998), Sebag (1996), Sebag and Rouveirol (1997, 2000) suggested to extend concept languages with logical disjunction only. This is due to the fact that conjunctive attributive languages were considered only which completeness increases with introducing logical disjunction. Thus, logical conjunction has never been taken into account.

version spaces are a partial solution to the problem of incomplete concept languages. Therefore, we conclude that the conjunctive version spaces can be used when the conjunctive extensions of concept languages are sufficiently complete with respect to the concept-learning tasks to be solved.

A next conclusion follows from definition 4.10. The definition states that conjunctive version spaces are version spaces in conjunctive extensions of concept languages. Thus, all properties, advantages and disadvantages of version spaces revealed in sections 3.3, 3.4 and 3.5 are inherited by conjunctive version spaces.

The second contribution of this chapter is that the conjunctive version-space abstract data type is expressed in terms of the version-space abstract data type[6]. The importance of the contribution is due to the fact that if the version-space abstract data type can be implemented, then the conjunctive version-space abstract data type can be implemented as well. That is why in the next two chapters we consider ways for implementing the version-space abstract data type for broad classes of concept languages.

By duality analogous contributions and conclusions can be obtained for disjunctive extensions of concept languages and disjunctive version spaces.

[6] Note that for the sake of simplification the operation $Converged?_{CVS}$ has been excluded from the set of the operations of the abstract data type of conjunctive version spaces. This is due to the fact that determining equivalence is an undecidable problem in general.

Chapter 5

Version-Space Representations and Algorithms

This chapter considers the problem of implementing the version-space abstract data type. In section 5.1 we show that implementing the version-space abstract data type for a concept language requires designing a version-space representation and algorithms of the basic version-space operations that are based on the representation. In this context we survey three version space representations together with their algorithms of the basic version-space operations. The survey starts in section 5.2 with list representation (Mitchell, 1978; Hirsh, 1992b). The representation is analysed and considered as inefficient. To design efficient version-space representations we employ the idea that concept languages have to be structured by a relation "more specific" (Plotkin, 1970; Vere, 1975; Mitchell, 1978). We define the relation in section 5.3 and derive the conditions when it is a partial ordering. This enables us to introduce three classes of structured concept languages, namely admissible, lower-admissible, and upper-admissible languages. The class of admissible concept languages is considered for the standard version-space representation by boundary sets (Mitchell, 1978). The representation is presented and analysed in section 5.4. The analysis is used in order to reveal the computational problem of the boundary sets that corresponds to the second part of our problem statement. The classes of lower-admissible and upper-admissible concept languages are considered for a version space representation called unilateral boundary sets (Hirsh, 1992b). The presentation is presented and analysed in section 5.4. It is shown that the unilateral boundary sets overcome partially the computational problem of the boundary sets.

The algorithms of the basic version-space operations for the boundary sets and the unilateral boundary sets are considered in detail. In this context the main contribution of this chapter is developing those algorithms that were not proposed in (Mitchell, 1978; Mitchell, 1982; Hirsh, 1992b).

The chapter ends in section 5.6. We analyse the presented version-space representations and we find that they are inefficient for the operation *Retraction*. Therefore, we conclude that we need a version-space representation that simultaneously solves the computational problem and the retraction problem; i.e., we need a version space representation that simultaneously solves the second and third part of our problem statement.

5.1 Terminology

Implementing the version-space abstract data type for a concept language requires (1) designing a version-space representation; and (2) designing algorithms of the basic version-space operations based on that representation. A version-space representation is a finite data structure that contains "information needed to reconstruct every concept description in the version space" (Mitchell, 1978). Given a concept language, a version-space representation can have two types of adequacy for the basic version-space operations (Hirsh, 1992b). The representation is *epistemologically* adequate if each operation has an algorithm based on the representation. The representation is *heuristically* adequate if the algorithm of each operation is tractable. An algorithm is tractable if its time complexity is polynomial in the size of the input and relevant properties of the concept language.

The heuristical adequacy of a representation for the basic version-space operations is closely related to the tractability of the representation. A version-space representation is tractable for a concept language if the representation can be computed for all possible training sets in time polynomial in their sizes and relevant properties of the language. We establish the tractability of a representation when it is heuristically adequate for the operations *Initialise* and *Update*, and its size is polynomial in the number of training instances and relevant properties of the concept language (Hirsh, 1992b).

Note that our definitions of tractable algorithms and tractable version-space representations are simplified. This is due to our assumption that the properties of the membership relation are a part of the properties of the concept language.

5.2 List Representation

The list representation is the most trivial version-space representation. It represents version spaces by listing all their elements. Hence, the definition of the representation coincides with definition 3.1 of version spaces.

In chapter 3 we have specified by version spaces all the basic version-space operations. Each of the operations is an algorithm if the concept languages used are finite. Thus, the list representation is epistemologically adequate with respect to the basic version-space operations for finite concept languages.

Hirsh (1992b) showed that the list representation is tractable and heuristically adequate for the basic version-space operations if the concept languages are small. A concept language is small if the representation can be computed for all possible training sets in time polynomial in their sizes and relevant properties of the language.

The condition for tractability and heuristical adequacy of the list representation for the basic version-space operations is not realistic. In general, version spaces can be infinite. Even if we consider them in finite concept languages the number of descriptions can be large so that the list representation is neither tractable nor heuristically adequate. Thus, we need version-space representations which sizes are not tied to the number of concept descriptions in version spaces (Mitchell, 1978).

5.3 Partially-Ordered Concept Languages

The key to find a good version-space representation is to observe that concept languages can be ordered. The order can be based on a relation "more specific". The relation is defined below.

5.3.1 Relation "More Specific"

The relation "more specific" orders the descriptions in concept languages according to the instances covered. It is taken from (Mitchell, 1978) and it is defined below.

Definition 5.1. (The relation "more specific" ($\leq$)) *Consider an instance language Li, a concept language Lc, and a membership relation M. Then:*

$$(\forall c_1, c_2 \in Lc)((c_1 \leq c_2) \leftrightarrow (\forall i \in Li)(M(c_1, i) \rightarrow M(c_2, i))).$$

The definition states that a description $c_1 \in Lc$ is more specific than a description $c_2 \in Lc$ ($c_1 \leq c_2$) if and only if for each instance $i \in Li$ if c_1 covers i then c_2 covers i as well. According to definition 2.7 it is equivalent to say that the extensional representation of the concept given by c_1 is a subset of the extensional representation of the concept given by c_2.

The relation "more specific" has a dual relation "more general". We express the relation "more general" by the relation "more specific": a description $c_1 \in Lc$ is more general than a description $c_2 \in Lc$ if and only if c_2 is more specific than c_1. Hence, when a description $c_1 \in Lc$ is more specific than another description $c_2 \in Lc$ we say that c_1 is a specialisation of c_2 and c_2 is a generalisation of c_1.

The reason why we consider the relation "$\leq$" is that it has one very important property: it is a partially-ordered relation. We prove this property in theorem 5.2 given below.

Theorem 5.2. *If the function $\mathcal{R}_{\mathfrak{C}}$ is injective, then the relation "$\leq$" is a partial ordering.*

Proof. (Adapted from (Mitchell, 1978)) In order to prove that the relation "$\leq$" is a partial ordering we have to prove that it is reflexive, anti-symmetric and transitive.

1). We prove that the relation "$\leq$" is reflexive. For an arbitrarily chosen $c \in Lc$:

$$(\forall i \in Li)(M(c,i) \rightarrow M(c,i)).$$

Thus, according to definition 5.1 it follows that:

$$c \leq c.$$

2). We prove that the relation "$\leq$" is anti-symmetric. Consider arbitrary chosen $c_1, c_2 \in Lc$ such that $c_1 \leq c_2$ and $c_2 \leq c_1$. According to definition 5.1 we have that:

$$(\forall i \in Li)(M(c_1,i) \rightarrow M(c_2,i))$$
$$(\forall i \in Li)(M(c_2,i) \rightarrow M(c_1,i)).$$

Thus,

$$(\forall i \in Li)(M(c_1,i) \leftrightarrow M(c_2,i)).$$

Using definition 3.11 this implies:

$$c_1 \sim c_2.$$

Thus, since $\mathcal{R}_{\mathfrak{C}}$ is injective by lemma 3.13 we have that:

$$c_1 = c_2.$$

3). We prove that the relation "$\leq$" is transitive. Consider arbitrarily chosen $c_1, c_2, c_3 \in Lc$ such that $c_1 \leq c_2$ and $c_2 \leq c_3$. According to definition 5.1 we have that:

$$(\forall i \in Li)(M(c_1,i) \rightarrow M(c_2,i))$$
$$(\forall i \in Li)(M(c_2,i) \rightarrow M(c_3,i)).$$

Thus,

$$(\forall i \in Li)(M(c_1,i) \rightarrow M(c_3,i)).$$

Using definition 5.1 this implies:

$$c_1 \leq c_3.$$

□

According to definition 2.10 the function $\mathcal{R}_{\mathfrak{C}}$ is injective. Hence, throughout the thesis we consider the relation "$\leq$" as partially ordered.

The relation "$\leq$" gives rise to a relation "$<$" that we call strict relation "more specific". The strict relation is introduced in definition 5.3 given below.

Definition 5.3. (The strict relation "more specific" ($<$)) *Consider a concept language Lc and the relation "$\leq$" defined on Lc. Then:*

$$(\forall c_1, c_2 \in Lc)((c_1 < c_2) \leftrightarrow ((c_1 \leq c_2) \wedge (c_1 \neq c_2))).$$

The definition states that a description $c_1 \in Lc$ is strictly more specific than a description $c_2 \in Lc$ ($c_1 < c_2$) if and only if c_1 is more specific than c_2 and c_1 is different from c_2. Therefore, the extensional representation of the concept given by c_1 is a strict subset of the extensional representation of the concept given by c_2.

In contrast with the relation "$\leq$" the relation "$<$" is not partially ordered. The relation is only transitive as shown in theorem 5.4.

Theorem 5.4. *The relation "$<$" is transitive.*

Proof. The proof is similar to that of the third part of theorem 5.2. □

Since the relations "$\leq$" and "$<$" are transitive they have two important properties:

(1) if a concept description c_1 is more specific than a concept description c_2, and the description c_1 covers all the instances of a subset $I \subseteq Li$, then the description c_2 covers all the instances of the subset I as well.

(2) if a concept description c_1 is more specific than a concept description c_2, and the description c_2 does not cover any instance from some subset $I \subseteq Li$, then the concept description c_1 does not cover any instance of the subset I as well.

The properties (1) and (2) are proven in lemmas 5.5 and 5.6, respectively.

Lemma 5.5. *Consider an instance language Li, a concept language Lc, a membership predicate M, and the relations "$\leq$" and "$<$" defined on Lc. If $c_1, c_2 \in Lc$ are such that $c_1 \leq c_2$ or $c_1 < c_2$, and if there exists $I \subseteq Li$ such that $(\forall i \in I)M(c_1, i)$ then:*

$$(\forall i \in I)M(c_2, i).$$

Proof. The proof follows from definitions 5.1 and 5.3. □

Lemma 5.6. *Consider an instance language Li, a concept language Lc, a membership predicate M, and the relations "$\leq$" and "$<$" defined on Lc. If $c_1, c_2 \in Lc$ are such that $c_1 \leq c_2$ or $c_1 < c_2$, and if there exists $I \subseteq Li$ such that $(\forall i \in I)\neg M(c_2, i)$ then:*

$$(\forall i \in I)\neg M(c_1, i).$$

Proof. The proof follows from definitions 5.1 and 5.3. □

5.3.2 Convexity, Boundedness, and Admissibility

If the relation "$\leq$" can be defined over a concept language Lc, then the language is a partially-ordered set. There exist two particular types of partially-ordered sets that we are interested in: convex sets and bounded sets.

Informally, a set C in a partially-ordered concept language Lc is convex if all descriptions in Lc between any two descriptions in C are also in C. This means that there are no holes in the set C. More formally, we have the following definition.

Definition 5.7. (Convex Sets) *Consider a concept language Lc and the relation "$\leq$" defined on Lc. A set $C \subseteq Lc$ is convex if and only if for all $c_1, c_2, c_3 \in Lc$ the conditions $c_1 \leq c_2 \leq c_3$ and $c_1, c_3 \in C$ imply that $c_2 \in C$.*

The reason why we consider convex sets is that version spaces are convex when they are defined in concept languages structured by the relation "$\leq$". This property of version spaces is proven in lemma 5.8.

Lemma 5.8. (Convexity of Version Spaces) *The version space VS of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ is convex if the relation "$\leq$" is defined on Lc.*

Proof. If the relation "$\leq$" is defined on Lc consider arbitrarily chosen $c_1, c_2, c_3 \in Lc$ such that $c_1 \leq c_2 \leq c_3$ and $c_1, c_3 \in VS$. To prove that VS is convex we have to show that $c_2 \in VS$ (see definition 5.7).

According to definition 3.1 $c_1 \in VS$ implies $(\forall i \in I^+)M(c_1, i)$. By lemma 5.5 formulas $(\forall i \in I^+)M(c_1, i)$ and $c_1 \leq c_2$ imply:

$$(\forall i \in I^+)M(c_2, i). \tag{5.1}$$

According to definition 3.1 $c_3 \in VS$ implies $(\forall i \in I^-)\neg M(c_3, i)$. By lemma 5.6 formulas $(\forall i \in I^-)\neg M(c_3, i)$ and $c_2 \leq c_3$ imply:

$$(\forall i \in I^-)\neg M(c_2, i). \tag{5.2}$$

According to definition 2.22 formulas (5.1) and (5.2) imply:

$$cons(c_2, \langle I^+, I^- \rangle).$$

Thus, since $c_2 \in Lc$ using definition 3.1 we conclude that:

$$c_2 \in VS.$$

□

The second type of partially-ordered sets that we are interested in are bounded sets. To consider them in detail we define the sets of minimal and maximal elements

of a partially-ordered set (Birkoff, 1973). A minimal element min of a partially-ordered set C is an element such that the inequality $c < min$ is not possible for any $c \in C$ (see definition 5.9 below). By duality, a maximal element max of a partially-ordered set C is an element such that the inequality $max < c$ is not possible for any $c \in C$ (see definition 5.10 below).

Definition 5.9. (Minimal Set) *If C is a partially-ordered set then:*

$$MIN(C) = \{c \in C | (\forall c' \in C) \neg (c' < c)\}.$$

Definition 5.10. (Maximal Set) *If C is a partially-ordered set then:*

$$MAX(C) = \{c \in C | (\forall c' \in C) \neg (c < c')\}.$$

Informally, a set C in a concept language Lc, structured by the relation "$\leq$", is bounded if all elements of C are between its minimal and maximal elements in the partially-ordered structure of Lc. More formally:

Definition 5.11. (Bounded Sets) *Consider a concept language Lc and the relation "$\leq$" defined on Lc. Then a set $C \subseteq Lc$ is bounded if and only if*

$$C \subseteq \{c \in Lc | (\exists s \in MIN(C))(\exists g \in MAX(C))((s \leq c) \wedge (c \leq g))\}.$$

The bounded sets are closely related to the lower-bounded and upper-bounded sets. Informally, a set C in a partially-ordered concept language Lc is lower-bounded if all elements of C are more general than its minimal elements in the partially ordered structure of Lc. More formally:

Definition 5.12. (Lower-Bounded Sets) *Consider a concept language Lc and the relation "$\leq$" defined on Lc. Then a set $C \subseteq Lc$ is lower-bounded if and only if:*

$$C \subseteq \{c \in Lc | (\exists s \in MIN(C))(s \leq c)\}.$$

By duality, a set C in a partially ordered concept language Lc is upper-bounded if all elements of C are more specific than its maximal elements in the partially-ordered structure of Lc. More formally:

Definition 5.13. (Upper-Bounded Sets) *Consider a concept language Lc and the relation "$\leq$" defined on Lc. Then a set $C \subseteq Lc$ is upper-bounded if and only if:*

$$C \subseteq \{c \in Lc | (\exists g \in MAX(C))(c \leq g)\}.$$

The three types of bounded sets give rise to three classes of admissible languages. We start with the bounded sets. They are used for defining a class of admissible concept languages given below.

Definition 5.14. (Admissible Concept Languages) *If the relation "$\leq$" is defined on a concept language Lc, then Lc is admissible if and only if every non-empty subset $C \subseteq Lc$ is bounded.*

The lower-bounded sets give rise to a class of lower-admissible concept languages.

Definition 5.15. (Lower-Admissible Concept Languages) *If the relation "$\leq$" is defined on a concept language Lc, then Lc is lower-admissible if and only if every non-empty subset $C \subseteq Lc$ is lower-bounded.*

By duality, the upper-bounded sets give rise to a class of upper-admissible concept languages.

Definition 5.16. (Upper-Admissible Concept Languages) *If the relation "$\leq$" is defined on a concept language Lc, then Lc is upper-admissible if and only if every non-empty subset $C \subseteq Lc$ is upper-bounded.*

5.4 Boundary Sets

The standard version-space representation is by boundary sets (Mitchell, 1978). It was proposed for admissible concept languages. Since every set is bounded in these languages, the boundary sets are defined to contain the minimal and maximal descriptions of version spaces. This implies that (1) they delimit version spaces in admissible concept languages, and (2) they are not tied to the number of concept descriptions in version spaces. Therefore, the boundary sets are a more efficient version-space representation than the list representation.

5.4.1 Definition and Correctness

Formally the boundary sets of version spaces are introduced in definition 5.17.

Definition 5.17. (Boundary Sets (BS)) *Given a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ with version space VS. If the concept language Lc is admissible, then VS is represented by an ordered pair $\langle S, G \rangle$ where:*

$$S = MIN(VS)$$
$$G = MAX(VS).$$

The set S is called the minimal boundary set of the version space VS; the set G is called the maximal boundary set of the version space VS. The sets completely specify the version space VS. The concept descriptions in VS are exactly those in the sets S and G, as well as all concept descriptions between these two sets in the partially-ordered structure of the concept language Lc. This property is proven in theorem 5.18 (Mitchell, 1978).

Theorem 5.18. (Correctness of Boundary Sets) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space* VS *and boundary sets* $\langle S, G \rangle$. *If the concept language* Lc *is admissible then:*

$$(\forall c \in Lc)((c \in VS) \leftrightarrow (\exists s \in S)(\exists g \in G)((s \leq c) \wedge (c \leq g))).$$

Proof. ($\rightarrow$) Since Lc is admissible, VS is bounded. Thus, according to definition 5.11:

$$VS \subseteq \{c \in Lc | (\exists s \in MIN(VS))(\exists g \in MAX(VS))((s \leq c) \wedge (c \leq g))\}.$$

But, by definition of boundary sets $S = MIN(VS)$ and $G = MAX(VS)$. Thus, for arbitrary chosen $c \in VS$ the desired implication holds:

$$(c \in VS) \rightarrow (\exists s \in S)(\exists g \in G)((s \leq c) \wedge (c \leq g)),$$

and the first part of the theorem is proven.

($\leftarrow$) Consider arbitrarily chosen $c \in Lc, s \in S$, and $g \in G$ such that $s \leq c \leq g$. Since Lc is admissible, the relation "$\leq$" is defined on Lc. By lemma 5.8 this implies that VS is convex. Thus, $s \leq c \leq g$ implies $c \in VS$, and the second part of the theorem is proven. □

The theorem reveals two principal features of the version-space boundary sets. The first one is that all the elements of version spaces are considered because the boundary sets correctly represent version spaces. The second feature is that all the elements of version spaces do not have to be explicitly enumerated due to the partial ordering relationship "$\leq$" defined over the concept languages used.

To compute and process the boundary sets by any algorithm, they need to be finite (Gunter, Ngair, and Subramanian, 1997). That is why we introduce constraint 5.19 given below.

Constraint 5.19. *Given an instance language* Li, *an admissible concept language* Lc, *and a membership* M, *the boundary sets* S *and* G *of the version space of every possible task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *are finite.*

We assume that the constraint holds for all the version space representations, considered in the thesis, that are partially or completely based on the minimal and maximal boundary sets.

Algorithm *Initialise*

Input: an admissible concept language Lc
Output: BS:$\langle S, G\rangle$ of a version space VS.

$S = MIN(Lc)$
$G = MAX(Lc)$
return $\langle S, G\rangle$.

Figure 5.1: The Algorithm of the Operation *Initialise*.

5.4.2 Algorithms of the Basic Version-Space Operations

This subsection presents algorithms of the basic version-space operations that are based on the boundary-set representation. The theorems for correctness of all the algorithms are given in Appendix A for the class of admissible concept languages. Therefore, all the algorithms presented in this section are given for admissible concept languages.

Note that the algorithms of the operation *Retraction* and *Subset?* are introduced for the first time. This is confirmed by the fact that the basic work of Mitchell (1978, 1997) on boundary sets did not consider these two operations. In addition to that we have to mention that the theorems for correctness of the algorithms of the operations *Converged?*, *Collapsed?* and *Classify* are formulated and proven for the first time as well, although the algorithms were introduced in (Mitchell, 1978).

5.4.2.1 Algorithm of the Operation *Initialise*

The operation *Initialise* sets the version space VS of a target concept when no training instances are available. The algorithm of the operation for the boundary sets is proposed for the class of admissible concept languages since boundary sets are a correct representation for this class.

The algorithm of the operation is presented in figure 5.1. Given an admissible concept language Lc, it initialises the boundary sets of a version space VS as follows: the minimal boundary set S is set equal to the minimal descriptions in the concept language Lc; the maximal boundary set G is set equal to the maximal descriptions in Lc. Thus, by theorem 5.18 the resulting boundary sets S and G represent the initial version space VS equal to the concept language Lc.

Algorithm *Update*

Input: i: a new training instance.
BS:$\langle S, G\rangle$ of a version space VS.
Output: BS:$\langle S', G'\rangle$ of a version space VS' s.t.:
if i is a positive instance $\mathfrak{p}$ then $VS' = \{c \in VS | M(c, \mathfrak{p})\}$;
if i is a negative instance $\mathfrak{n}$ then $VS' = \{c \in VS | \neg M(c, \mathfrak{n})\}$.

if instance i is a positive instance $\mathfrak{p}$ **then**
$S' = MIN(\{c \in Lc | (\exists s \in S)(\exists g \in G)((s \leq c) \wedge (c \leq g)) \wedge M(c, \mathfrak{p})\})$
$G' = \{g \in G | M(g, \mathfrak{p})\}$
if instance i is a negative instance $\mathfrak{n}$ **then**
$G' = MAX(\{c \in Lc | (\exists s \in S)(\exists g \in G)((s \leq c) \wedge (c \leq g)) \wedge \neg M(c, \mathfrak{n})\})$
$S' = \{s \in S | \neg M(s, \mathfrak{n})\}$
return $\langle S', G'\rangle$.

Figure 5.2: The Algorithm of the Operation *Update*.

5.4.2.2 Algorithm of the Operation *Update*

The operation *Update* revises the version space of a target concept when a new instance is added to the training sets. The algorithm of the operation for the boundary sets has two procedures for handling positive and negative training instances, respectively. The procedures are based on theorems A.1 and A.2 (see Appendix A). The theorems are correct for the class of admissible concept languages. Thus, the algorithm is correct for this class as well.

The algorithm is presented in figure 5.2. Given a new positive training instance $\mathfrak{p}$ it forms the boundary sets S' and G' of the new updated version space VS' from the boundary sets S and G of the initial version space VS. The set S' is formed from those minimal generalisations of the elements of the set S that (1) are covered by at least one element of the set G; and (2) cover the instance $\mathfrak{p}$. The set G' is formed from those elements of the set G that cover the instance $\mathfrak{p}$. Thus, by theorem A.1 the resulting boundary sets S' and G' correctly represent the version space VS'.

The algorithm's behaviour for a negative instance is dual to that for a positive instance; hence it is analogous in form.

5.4.2.3 Algorithm of the Operation *Retraction*

The operation *Retraction* revises the version space of a target concept when an instance is removed from the training sets. The algorithm of the operation for the boundary sets has two procedures for handling positive and negative training

Algorithm *Retraction*
Input: i: a training instance to be retracted.
BS:$\langle S, G\rangle$ of a version space VS.
Lc: a concept language.
I^+: a set of positive training instances.
I^-: a set of negative training instances.
Output:
BS:$\langle S', G'\rangle$ of $VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$
if i is a positive instance $\mathfrak{p}$.
BS:$\langle S', G'\rangle$ of $VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\}$
if i is a negative instance $\mathfrak{n}$.

if instance i is a positive instance $\mathfrak{p}$ **then**
$S' = MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}))$
$G' = MAX(G \cup MAX(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}))$
if instance i is a negative instance $\mathfrak{n}$ **then**
$S' = MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\}))$
$G' = MAX(G \cup MAX(\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\}))$
return $\langle S', G'\rangle$.

Figure 5.3: The Algorithm of the Operation *Retraction*.

instances, respectively. The procedures are based on theorems A.5 and A.6 (see Appendix A). The theorems are correct for the class of admissible concept languages. Thus, the algorithm is correct for this class as well.

The algorithm is given in figure 5.3. When a positive training instance $\mathfrak{p}$ is retracted the algorithm forms the boundary sets S' and G' of the new retracted version space VS' from the boundary sets S and G of the initial version space VS, the concept language Lc used, and the training sets I^+ and I^-. This is realised in two steps. In the first step the algorithm forms the minimal boundary set S'. It is set equal to the minimal elements of the union of the minimal boundary set S and the minimal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$. In the second step the algorithm forms the maximal boundary set G'. It is set equal to the maximal elements of the union of the maximal boundary set G and the maximal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$. Thus, by theorem A.5 the resulting boundary sets S' and G' correctly represent the version space VS'.

The algorithm's behaviour for a negative instance is dual to that for a positive instance; hence it is analogous in form.

The main problem of the algorithm of the operation *Retraction* is how to form

the boundary sets of the version spaces $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ and $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{n}\}, I^- \cup \{\mathfrak{n}\}\rangle)\}$. The general solution is to form these boundary sets by consequently applying the operation *Update*.

5.4.2.4 Algorithm of the Operation *Collapsed?*

The operation *Collapsed?* determines whether a version space is empty. The algorithm of the operation is based on theorems A.7 and A.8 (see Appendix A). Since the theorems are correct for the class of admissible concept languages, the algorithm is correct for this class as well.

The algorithm checks for a collapse a version space VS by checking either the minimal boundary set S or the maximal boundary set G of VS. If the chosen set is not empty then by theorem A.7 or A.8 the version space VS is not empty and the algorithm returns false. Otherwise, it returns true; i.e., the version space VS is empty.

5.4.2.5 Algorithm of the Operation *Converged?*

The operation *Converged?* determines whether a version space has only one element[1]. The algorithm of the operation is based on theorem A.9 (see Appendix A). Since the theorem is correct for the class of admissible concept languages, the algorithm is correct for this class as well.

The algorithm tests a version space VS for convergence in two steps. In the first step it checks whether VS is empty using the algorithm of the operation *Collapsed?*. If so, then according to definition 3.12 VS is not converged, and the algorithm returns false. Otherwise, the algorithm executes the second step. It checks the sizes of the minimal and maximal boundary sets S and G of VS. If the sizes are equal to one then the algorithm checks whether the boundary sets are equal. If so, by theorem A.9 VS has only one element and the algorithm returns true. Otherwise, VS has more than one element and the algorithm returns false.

5.4.2.6 Algorithm of the Operation *Classify*

The operation *Classify* determines the classification of an instance using the rule of the unanimous vote of descriptions in version spaces. The algorithm of the operation is based on theorems A.10 and A.11 (see Appendix A). Since the theorems are correct for the class of admissible concept languages, the algorithm is correct for this class as well.

The algorithm of the operation *Classify* is given in figure 5.4. If an instance has to be classified by a version space VS, then the algorithm checks whether VS is empty using the algorithm of the operation *Collapsed?*. If so, then it returns

[1] Note that the algorithm is given under assumption that the function $\mathcal{R}_{\mathfrak{C}}$ is injective. By lemma 3.13 this means that equivalent descriptions are equal.

Algorithm *Classify*

Input: i: a instance to be classified.
BS:$\langle S, G\rangle$ of a version space VS.
Output: "+" if $(VS \neq \emptyset) \wedge (\forall c \in VS) M(c, i)$.
"−" if $(VS \neq \emptyset) \wedge (\forall c \in VS) \neg M(c, i)$.
"?" otherwise.

if *Collapsed?*$(\langle S, G\rangle)$ **then**
return "?"
if $(\forall s \in S) M(s, i)$ **then**
return "+"
if $(\forall g \in G) \neg M(g, i)$ **then**
return "−"
return "?".

Figure 5.4: The Algorithm of the Operation *Classify*.

"?" (see definition 3.14). Otherwise, the algorithm checks whether the instance is covered by all the descriptions of the minimal boundary set S. If so, then by theorem A.10 all the descriptions of the version space VS cover the instance and the algorithm returns "+". Otherwise, the algorithm checks whether the instance is not covered by any description of the maximal boundary set G. If so, then by theorem A.11 all the descriptions of the version space VS do not cover the instance and the algorithm returns "−". If positive and negative classification of the instance cannot be determined, the algorithm returns "?".

5.4.2.7 Algorithm of the Operation *Member?*

The operation *Member?* determines whether a concept description belongs to a version space VS. The algorithm of the operation is based on theorem 5.18. Since the theorem is correct for the class of admissible concept languages, the algorithm is correct for this class as well.

When a concept description c has to be checked for a membership to the version space VS the algorithm tests:

(1) the existence of at least one element of the minimal boundary set that is more specific than c; and

(2) the existence of at least one element of the maximal boundary set that is more general than c.

Algorithm *Intersection*

Input: BS:$\langle S_1, G_1\rangle$ of a version space VS_1.
BS:$\langle S_2, G_2\rangle$ of a version space VS_2.
Output: BS:$\langle S_{12}, G_{12}\rangle$ of a version space $VS_{12} = VS_1 \cap VS_2$.

$S_{12} = MIN(\{c \in Lc | (\exists s_1 \in S_1)(\exists s_2 \in S_2)((s_1 \leq c) \wedge (s_2 \leq c))\})$
$S_{12} = \{s \in S_{12} | (\exists g_1 \in G_1)(\exists g_2 \in G_2)((s \leq g_1) \wedge (s \leq g_2))\}$

$G_{12} = MAX(\{c \in Lc | (\exists g_1 \in G_1)(\exists g_2 \in G_2)((c \leq g_1) \wedge (c \leq g_2))\})$
$G_{12} = \{g \in G_{12} | (\exists s_1 \in S_1)(\exists s_2 \in S_2)((s_1 \leq g) \wedge (s_2 \leq g))\}$
return $\langle S_{12}, G_{12}\rangle$.

Figure 5.5: The Algorithm of the Operation *Intersection*.

If the conditions (1) and (2) do hold, then by theorem 5.18 the description c belongs to the version space VS and the algorithm returns true. Otherwise, the description c does not belong to the version space VS and the algorithm returns false.

5.4.2.8 Algorithm of the Operation *Intersection*

The operation *Intersection* computes a version space VS_{12} that is the intersection of two other version spaces VS_1 and VS_2. The algorithm of the operation is based on theorem A.12 (see Appendix A)[2]. Since the theorem is correct for the class of admissible concept languages, the algorithm is correct for this class as well.

The algorithm of the operation *Intersection* is given in figure 5.5. When version space VS_1 and VS_2 have to be intersected, it computes the boundary sets S_{12} and G_{12} of the intersected version space VS_{12} as follows.

The set S_{12} is formed in two steps. In the first step the set is initialised equal to the set of minimal generalisations of the elements of the minimal boundary sets S_1 and S_2. In the second step the set S_{12} is pruned: all the elements that are not more specific than at least one combination of the elements of the maximal boundary sets G_1 and G_2 are removed.

By duality the set G_{12} is formed in two steps as well. In the first step the set is initialised equal to the set of maximal specialisations of the elements of the maximal boundary sets G_1 and G_2. In the second step the set G_{12} is pruned: all the elements that are not more general than at least one combination of the elements of the minimal boundary sets S_1 and S_2 are removed.

[2] Theorem A.12 has been proven in (Hirsh, 1989).

Algorithm *Subset?*

Input: BS:$\langle S_1, G_1 \rangle$ of a version space VS_1.
BS:$\langle S_2, G_2 \rangle$ of a version space VS_2.
Output: true if $(VS_1 \subseteq VS_2)$.
false if $\neg(VS_1 \subseteq VS_2)$.

if $(\exists s_1 \in S_1)(\forall s_2 \in S_2)\neg(s_2 \leq s_1)$ **then**
return false
if $(\exists g_1 \in G_1)(\forall g_2 \in G2)\neg(g_1 \leq g_2)$ **then**
return false
return true.

Figure 5.6: The Algorithm of the Operation *Subset?*.

5.4.2.9 Algorithm of the Operation *Subset?*

The operation *Subset?* determines whether a version space VS_1 is a subset of another version space VS_2. The algorithm of the operation is based on theorem A.13 (see Appendix A). Since theorem A.13 is correct for the class of admissible concept languages, the algorithm is correct for this class as well.

The algorithm of the operation *Subset?* is given in figure 5.6. Its input are boundary sets $\langle S_1, G_1 \rangle$ and $\langle S_2, G_2 \rangle$ of two version spaces VS_1 and VS_2. To test whether VS_1 is a subset of VS_2 the algorithm checks the existence of at least one element $s_1 \in S_1$ that is not more general than any element of the set S_2. If so, by theorem A.13 VS_1 is not a subset of VS_2 and the algorithm returns false. Otherwise, the algorithm checks the existence of at least one element $g_1 \in G_1$ that is not more specific than any element of the set G_2. If so, by theorem A.13 VS_1 is not a subset of VS_2 and the algorithm returns false. Otherwise, it returns true: VS_1 is a subset of VS_2.

5.4.2.10 Algorithm of the Operation *Equal?*

The operation *Equal?* determines whether a version space VS_1 is equal to another version space VS_2. The algorithm of the operation is based on theorem A.14 (see Appendix A). Since the theorem is correct for the class of admissible concept languages, the algorithm is correct for this class as well.

When two version spaces have to be checked for equality the algorithm compares their boundary sets. If the boundary sets are equal, then by theorem A.14 the version spaces are equal and the algorithm returns true. Otherwise, it returns false: version spaces are not equal.

5.4.3 Worst-Case Complexity Analysis

This subsection presents a worst-case complexity analysis of the boundary-set algorithms of the basic version-space operations (Hirsh, 1992a). The analysis is made under two worst-case assumptions:

(1) the elements of the minimal boundary sets do not cover any positive and any negative training instance;

(2) the elements of the maximal boundary sets do cover every positive and every negative training instance.

The analysis is made in terms of:

- the number P of positive instances;
- the number N of negative instances;
- the largest size Σ of the set $MIN(Lc)$;
- the largest size Γ of the set $MAX(Lc)$;
- the worst-case time complexity $t^{\downarrow}$ for generating the set $MIN(Lc)$;
- the worst-case time complexity $t^{\uparrow}$ for generating the set $MAX(Lc)$;
- the maximal number $\hat{s}$ of minimal specialisations of a concept description;
- the maximal number $\check{g}$ of minimal generalisations of a concept description;
- the worst-case time complexity t_s to determine all the minimal specialisations of a concept description;
- the worst-case time complexity t_g to determine all the minimal generalisations of a concept description;
- the worst-case time complexity t_c to test whether one concept description covers another concept description or an instance.

5.4.3.1 Worst-Case Space-Complexity Analysis of the Representation

In the worst case the minimal boundary set S has $\Sigma\check{g}^P$ elements and the maximal boundary set G has $\Gamma\hat{s}^N$ elements[3]. Thus, the size of the boundary sets in the worst-case is equal to $O(\Sigma\check{g}^P + \Gamma\hat{s}^N)$.

[3] This can be proven by induction on the sizes of the numbers P and N.

5.4.3.2 Worst-Case Time-Complexity Analysis of the Algorithms

This subsection presents a worst-case time complexity analysis of the algorithms of the basic version-space operations based on boundary sets.

The Algorithm of the Operation *Initialise*. The worst-case time complexity of the algorithm is equal to the time complexity $t^{\downarrow}$ for generating the minimal boundary set S plus the time complexity $t^{\uparrow}$ for generating the maximal boundary set G (see figure 5.1). Thus, the worst-case time complexity of the algorithm is $O(t^{\downarrow} + t^{\uparrow})$.

The Algorithm of the Operation *Update*. The algorithm consists of two procedures for handling positive and negative training instances, respectively. Hence, the time complexity analysis is realised for each of the procedures separately.

The worst-case time complexity of the procedure for processing a positive training instance is equal to $O(|S|t_g + |S|\check{g}t_c + |S||G|\check{g}t_c + (|S|\check{g})^2 t_c + |G|t_c)$ (see figure 5.2). The term $O(|S|t_g)$ arises because each element of the minimal boundary set S has to be minimally generalised to cover the instance. The term $O(|S|\check{g}t_c)$ arises because $|S|\check{g}$ minimal generalisations are checked whether they cover the positive instance. The term $O(|S||G|\check{g}t_c)$ arises because $|S|\check{g}$ minimal generalisations are compared with the elements of the set G. The term $O((|S|\check{g})^2 t_c)$ arises because each two minimal generalisations are compared so that the non-minimal elements are removed. The term $O(|G|t_c)$ arises because the elements of the set G are checked whether they cover the instance. Thus, the overall worst-case time complexity is equal to $O(|S|t_g + |S||G|\check{g}t_c + (|S|\check{g})^2 t_c)$.

The worst-case time complexity of the procedure for processing one negative instance is derived by duality. It is $O(|G|t_s + |S||G|\hat{s}t_c + (|G|\hat{s})^2 t_c)$.

The Algorithm of the Operation *Retraction*. The algorithm consists of two procedures for handling positive and negative training instances, respectively. Thus, the time complexity analysis is realised for each of the procedures separately.

The procedure for handling one positive instance generates the minimal and maximal boundary sets of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ (see figure 5.3). The worst-case sizes of these minimal and maximal boundary sets are $O(\Sigma\check{g}^{P-1})$ and $O(\Gamma\hat{s}^{N+1})$, respectively. We substitute these sizes in the worst-case time-complexity expressions of the algorithm of the operation *Update*. Thus, after $P-1$ positive and $N+1$ negative training instances:

- the worst-case time complexity of the procedure of the operation *Update* given a positive instance is $O(\Sigma\check{g}^{P-1}t_g + \Sigma\check{g}^{P-1}\Gamma\hat{s}^{N+1}\check{g}t_c + (\Sigma\check{g}^{P-1}\check{g})^2 t_c)$;
- the worst-case time complexity of the procedure of the operation *Update* given a negative instance is $O(\Gamma\hat{s}^{N+1}t_s + \Sigma\check{g}^{P-1}\Gamma\hat{s}^{N+1}\hat{s}t_c + (\Gamma\hat{s}^{N+1}\hat{s})^2 t_c)$.

To generate the minimal and maximal boundary sets of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ we need to apply the algorithm of the operation *Initialise*, $P-1$ times the procedure of the operation *Update* given a positive

instance, and $N+1$ times the procedure of the operation *Update* given a negative instance. Thus, the worst-case time complexity for generating these sets is $O(t^{\downarrow}+t^{\uparrow}+(\Sigma\check{g}^{P-1}t_g+\Sigma\check{g}^{P-1}\Gamma\hat{s}^{N+1}\check{g}t_c+(\Sigma\check{g}^{P-1}\check{g})^2t_c)+(\Gamma\hat{s}^{N+1}t_s+\Sigma\check{g}^{P-1}\Gamma\hat{s}^{N+1}\hat{s}t_c+(\Gamma\hat{s}^{N+1}\hat{s})^2t_c))$[4]. After a simplification it is $O(t^{\downarrow}+t^{\uparrow}+\Sigma\check{g}^{P-1}t_g+\Sigma\check{g}^{P}\Gamma\hat{s}^{N+1}t_c+\Sigma^2\check{g}^{2P}t_c+\Gamma\hat{s}^{N+1}t_s+\Sigma\check{g}^{P-1}\Gamma\hat{s}^{N+2}t_c+\Gamma^2\hat{s}^{2(N+2)}t_c)$. The elements of the minimal and maximal boundary sets of the version space $\{c\in Lc|cons(c,\langle I^+-\{\mathfrak{p}\},I^-\cup\{\mathfrak{p}\}\rangle)\}$ are added to the minimal and maximal boundary sets S and G of the resulting version space. Therefore, each two elements of the set S are compared so that the non-minimal elements are removed, and each two elements of the set G are compared so that the non-maximal elements are removed. The worst-case time complexities of these two calculations are $O(|S|\Sigma\check{g}^{P-1}t_c)$ and $O(|G|\Gamma\hat{s}^{N+1}t_c)$, respectively. Thus, worst-case time complexity of the procedure for handling one positive instance is $O(t^{\downarrow}+t^{\uparrow}+\Sigma\check{g}^{P-1}t_g+\Sigma\check{g}^{P}\Gamma\hat{s}^{N+1}t_c+\Sigma^2\check{g}^{2P}t_c+\Gamma\hat{s}^{N+1}t_s+\Sigma\check{g}^{P-1}\Gamma\hat{s}^{N+2}t_c+\Gamma^2\hat{s}^{2(N+2)}t_c+|S|\Sigma\check{g}^{P-1}t_c+|G|\Gamma\hat{s}^{N+1}t_c)$.

The worst-case time complexity of the procedure for handling one negative instance is derived by analogy. It is $O(t^{\downarrow}+t^{\uparrow}+\Sigma\check{g}^{P+1}t_g+\Sigma\check{g}^{P+2}\Gamma\hat{s}^{N-1}t_c+\Sigma^2\check{g}^{2(P+2)}t_c+\Gamma\hat{s}^{N-1}t_s+\Sigma\check{g}^{P+1}\Gamma\hat{s}^{N}t_c+\Gamma^2\hat{s}^{2N}t_c+|S|\Sigma\check{g}^{P+1}t_c+|G|\Gamma\hat{s}^{N-1}t_c)$.

The Algorithm of the Operation *Collapsed?*. The worst-case time complexity of the algorithm is $O(1)$. This is due to the fact that the algorithm checks whether exactly one boundary set is empty.

The Algorithm of the Operation *Converged?*. The worst-case time complexity of the algorithm is equal to $O(1+1+t_c)$. The first term $O(1)$ is the worst-case time complexity of the algorithm of operation *Collapsed?*. The second term $O(1)$ is the complexity for checking whether each of the boundary sets consists of exactly one element. The term $O(t_c)$ is the complexity for checking whether the only elements of the boundary sets are equal. Thus, the overall worst-case time complexity of the algorithm of the operation *Converged?* is equal to $O(t_c)$.

The Algorithm of the Operation *Classify*. The worst-case time complexity of the positive classification is $O(|S|t_c)$ since all the elements of the minimal boundary set S are checked whether they cover the instance to be classified (see figure 5.4). The time complexity of the negative classification is $O(|G|t_c)$ since all the elements of the maximal boundary set G are checked whether they reject the instance to be classified.

The worst-case time complexity of the algorithm of the operation *Classify* is equal to the sum of the worst-case time complexity of the algorithm of the operation *Collapsed?*, the worst-case time complexity of the positive classification, and the worst-case time complexity of the negative classification. Thus, it is equal to $O(t_c+|S|t_c+|G|t_c)=O(|S|t_c+|G|t_c)$.

The Algorithm of the Operation *Member?*. The worst-case time complexity of the algorithm is $O(|S|t_c+|G|t_c)$. The term $O(|S|t_c)$ arises because all the descriptions

[4] Note that only the worst-case time complexities of the procedures of the operation *Update* after $P-1$ and $N+1$ have an importance for forming this complexity.

of the minimal boundary set S are checked whether they are more specific than a given concept description c. The term $O(|G|t_c)$ arises because all the descriptions of the maximal boundary set G are checked whether they are more general than c.
The Algorithm of the Operation *Intersection*. The worst-case time complexity of the algorithm is equal to the sum of the time complexities for generating the minimal and maximal boundary sets S_{12} and G_{12} (see figure 5.5). The time complexity for generating the set S_{12} is $O(2|S|t_g + (|S|\check{g})^2 t_c + (|S|\check{g})^2 t_c + 2|S||G|t_c)$, where S (G) is the largest set among the sets S_1 (G_1) and S_2 (G_2). The term $O(2|S|t_g)$ arises because the elements of both sets S_1 and S_2 have to be minimally generalised. The first term $O((|S|\check{g})^2 t_c)$ arises because $|S|\check{g}$ minimal generalisations of the set S_1 are compared with $|S|\check{g}$ minimal generalisations of the set S_2 in order to determine common minimal generalisations. The second term $O((|S|\check{g})^2 t_c)$ arises because each two minimal generalisations are compared so that the non-minimal elements are removed. The term $O(2|S||G|t_c)$ arises because each minimal generalisation of the set S_{12} is compared with the elements of the sets G_1 and G_2. Thus, the overall complexity for generating the set S_{12} is $O(|S|t_g + (|S|\check{g})^2 t_c + |S||G|t_c)$.

The worst-case time complexity for generating the set G_{12} can be derived by duality and it is $O(|G|t_s + (|G|\hat{s})^2 t_c + |S||G|t_c)$. Thus, the worst-case time complexity of the intersection algorithm is $O(|S|t_g + |G|t_s + (|S|\check{g})^2 t_c + (|G|\hat{s})^2 t_c + |S||G|t_c)$.
The Algorithm of the Operation *Subset?*. The worst-case time complexity of the algorithm is equal to $O(|S|^2 t_c + |G|^2 t_c)$, where S (G) is the largest set among the sets S_1 (G_1) and S_2 (G_2) (see figure 5.6). The term $O(|S|^2 t_c)$ is because the elements of the minimal boundary set S_1 are compared with the elements of the minimal boundary set S_2. The term $O(|G|^2 t_c)$ can be explained by duality.
Time Complexity of the Algorithm of the Operation *Equal?*. The worst-case time complexity of the algorithm is derived analogously to that of the algorithm of the operation *Subset?* and it is equal to $O(|S|^2 t_c + |G|^2 t_c)$, where S (G) is the largest set among the sets S_1 (G_1) and S_2 (G_2).

5.4.4 Adequacy of the Representation

In this subsection we investigate the tractability and the epistemological and heuristical adequacy of the boundary-set representation for the basic version-space operations.

5.4.4.1 Epistemological Adequacy for the Basic Version-Space Operations

Subsection 5.4.2 has presented the boundary-set algorithms of all the basic version-space operations. The correctness of the algorithms has been proven for admissible languages (see (Mitchell, 1978) and Appendix A). Thus, *the boundary-set representation is epistemologically adequate for all the basic version-space operations for the class of admissible concept languages.*

5.4.4.2 Heuristical Adequacy for the Basic Version-Space Operations

We investigate the heuristical adequacy of the boundary sets for the class of admissible concept languages. We use the results of the worst-case complexity analysis from subsection 5.4.3. They show that the time complexities of the algorithms of the basic version-space operations *Initialise, Update, Collapsed?, Converged?, Classify, Member?, Intersection, Subset?*, and *Equal?* are polynomial in the sizes of the input $|S|$ and $|G|$, the complexities $t^{\downarrow}$, $t^{\uparrow}$, t_c, t_s, t_g, and the maximal numbers $\hat{s}$ and $\check{g}$. Thus, the algorithms of these operations are tractable; i.e., the boundary-set representation is heuristically adequate for these operations for the class of admissible concept languages, if the complexities $t^{\downarrow}$, $t^{\uparrow}$, t_c, t_s, t_g, and the maximal numbers $\hat{s}$ and $\check{g}$ are polynomial in relevant properties of these concept languages.

The algorithm of the basic version-space operation *Retraction* requires additional attention. The time complexity of the algorithm is polynomial in the size of the input $|S|$ and $|G|$, the complexities $t^{\downarrow}$, $t^{\uparrow}$, t_c, t_s, t_g, but it is exponential in the numbers P and N of positive and negative training instances with bases the maximal numbers $\hat{s}$ and $\check{g}$, respectively. Thus, in general the algorithm of the operation *Retraction* is intractable. Nevertheless, for a particular case if (1) the complexities $t^{\downarrow}$, $t^{\uparrow}$, t_c, t_s, t_g are polynomial in relevant properties of the concept languages used, and (2) the maximal numbers $\hat{s}$ and $\check{g}$ are equal to one, the algorithm is tractable.

5.4.4.3 Tractability

We investigate the tractability of the boundary-set representation for the class of admissible concept languages. According to the terminology in section 5.1 the boundary sets are tractable for a concept language when they are heuristically adequate for the operations *Initialise* and *Update*, and their size is polynomial in the number of training instances and relevant language properties. In this context we have established that:

- the boundary sets are heuristically adequate for the operation *Initialise* if the generation complexities $t^{\downarrow}$ and $t^{\uparrow}$ are polynomial in relevant properties of the concept language used;
- the boundary sets are heuristically adequate for the operation *Update* if the complexities t_c, t_s, t_g, and the maximal numbers $\hat{s}$ and $\check{g}$ are polynomial in relevant properties of the concept language used;
- the size of the boundary sets is exponential in the numbers P and N of positive and negative training instances with bases the maximal numbers $\hat{s}$ and $\check{g}$, respectively.

Thus, in general the boundary-set representation is intractable for the class of admissible concept languages. Nevertheless, for a particular case if (1) the complexities $t^{\downarrow}$, $t^{\uparrow}$, t_c, t_s, t_g are polynomial in relevant properties of the concept languages

used, and (2) the maximal numbers $\hat{s}$ and $\check{g}$ are equal to one, the representation is tractable.

The intractability of the boundary sets has been discussed in chapter 1. It has been called the computational problem of the boundary sets. Since it is a part of our problem statement, in the next subsection we illustrate this problem for a particular class of concept languages.

5.4.5 Haussler's Example of Boundary Sets

Haussler (1988) showed that for simple 1-CNF languages with Boolean attributes the sizes of the maximal boundary sets can grow exponentially in the number of negative training instances. We consider this example of the boundary sets as follows. Let the instance language Li and the concept language Lc be 1-CNF languages with n Boolean attributes such that n is even and the single representation trick holds; i.e., $Li \subseteq Lc$ (Cohen and Feigenbaum, 1981). Let the first positive instance be:

$$+ (1,1,1,1,1,1,\cdots,1,1)$$

followed by the $\frac{1}{2}n$ negative instances:

$$- (0,0,1,1,1,1,\cdots,1,1)$$
$$- (1,1,0,0,1,1,\cdots,1,1)$$
$$- (1,1,1,1,0,0,\cdots,1,1)$$
$$\cdots$$
$$- (1,1,1,1,1,1,\cdots,0,0)$$

After assimilating the instances the maximal boundary set G can be represented as follows:

$$((1,?,?,?,?,?,\cdots,?,?) \text{ OR } (?,1,?,?,?,?,\cdots,?,?)) \text{ AND}$$
$$((?,?,1,?,?,?,\cdots,?,?) \text{ OR } (?,?,?,1,?,?,\cdots,?,?)) \text{ AND}$$
$$((?,?,?,?,1,?,\cdots,?,?) \text{ OR } (?,?,?,?,?,1,\cdots,?,?)) \text{ AND}$$
$$\cdots$$
$$((?,?,?,?,?,?,\cdots,1,?) \text{ OR } (?,?,?,?,?,?,\cdots,?,1))$$

From this expression it is easy to see that if the set G is given in 1-CNF language with Boolean attributes then it will have $2^{\frac{n}{2}}$ elements. Hence, the size of the maximal boundary set G grows exponentially in the number of negative instances.

Hirsh *et al.* (1997) found the same effect for a dual case: the sizes of the minimal boundary sets can grow exponentially in the number of positive training instances for 1-DNF languages with Boolean attributes.

5.5 Unilateral Boundary Sets

To overcome the computational problem of the boundary sets Hirsh (1992b) proposed two dual alternative representations of version spaces. The first one consists of the minimal boundary set and the set of negative training instances. Therefore, the representation can be considered as unilateral boundary sets. It can be applied when:

- the concept languages are admissible and the maximal boundary sets grow exponentially in the number of negative instances; or
- the concept languages are lower-admissible and the maximal boundary sets do not exist.

By duality the second representation consists of the set of positive training instances and the maximal boundary set. Therefore, the representation is unilateral boundary sets as well. It can be applied when:

- the concept languages are admissible and the minimal boundary sets grow exponentially in the number of positive instances; or
- the concept languages are upper-admissible and the minimal boundary sets do not exist.

Note that the unilateral-boundary-set representations are not tied to the number of concept descriptions in version spaces. Hence, they are more efficient than the list representation. Moreover, both unilateral boundary sets are a solution to the computational problem of the boundary sets. But, the solution is partial. This is due to the fact each of the representations contains exactly one boundary set and there is no guarantee that the set does not grow exponentially in the number of training instances.

In the rest of this section we consider the unilateral boundary sets based on the minimal boundary set. This is due to the fact that both unilateral boundary sets are mutually dual.

5.5.1 Definition and Correctness

The unilateral boundary sets based on the minimal boundary set S are formally introduced in definition 5.17 below.

Definition 5.20. (Unilateral Boundary Sets based on the set S (UBSS)) *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ with version space VS. If the concept language Lc is lower-admissible, then VS is represented by an ordered pair $\langle S, I^- \rangle$ where:*

$$S = MIN(VS).$$

As usually the set S is called the minimal boundary set of the version space VS; the set I^- is the set of negative training instances. The sets completely specify the version space VS. The concept descriptions in VS are exactly those in the set S, as well as all concept descriptions in the partially-ordered structure of the concept language used that (1) are more general than the elements of the set S, and (2) do not cover any instance from the set I^-. This property is proven in theorem 5.21.

Theorem 5.21. (Correctness of UBSS) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS represented by an ordered pair* $\langle S, I^- \rangle$. *If the concept language Lc is lower-admissible then:*

$$(\forall c \in Lc)((c \in VS) \leftrightarrow ((\exists s \in S)(s \leq c) \wedge (\forall n \in I^-)\neg M(c,n))).$$

Proof. ($\rightarrow$) Consider an arbitrary $c \in VS$. By definition of version spaces $cons(c, \langle I^+, I^- \rangle)$. Thus, according to definition 2.22:

$$(\forall n \in I^-)\neg M(c,n). \tag{5.3}$$

Since Lc is lower-admissible, VS is lower-bounded. Thus, according to definition 5.12:

$$VS \subseteq \{c \in Lc | (\exists s \in MIN(VS))(s \leq c)\}.$$

But, by definition of UBSS $S = MIN(VS)$. Thus,

$$(\exists s \in S)(s \leq c). \tag{5.4}$$

Combining (5.3) and (5.4) we have that:

$$(\exists s \in S)(s \leq c) \wedge (\forall n \in I^-)\neg M(c,n),$$

and the first part of the theorem is proven.

($\leftarrow$) Consider an arbitrarily chosen $c \in Lc$ and $s \in S$ such that:

$$(s \leq c) \tag{5.5}$$
$$(\forall n \in I^-)\neg M(c,n). \tag{5.6}$$

Since $s \in S$ and $S \subseteq VS$, by definition of version spaces $cons(s, \langle I^+, I^- \rangle)$. Thus, according to definition 2.22:

$$(\forall p \in I^+)M(s,p). \tag{5.7}$$

By lemma 5.5 formulas (5.5) and (5.7) imply:

$$(\forall p \in I^+)M(c,p). \tag{5.8}$$

According to definition 2.22 formulas (5.6) and (5.8) are equivalent to:

$$cons(c, \langle I^+, I^- \rangle). \tag{5.9}$$

Thus, using definition 3.1 $c \in Lc$ and (5.9) imply that $c \in VS$, and the second part of the theorem is proven. □

By theorem 5.21 the UBSS representation preserves two principal features of the boundary sets: all the elements of version spaces are considered and this does not require their explicit enumeration.

To compute and process the UBSS representation of version spaces by any algorithm, the minimal boundary set needs to be finite (Gunter *et al.*, 1997). That is why we introduce constraint 5.22 below.

Constraint 5.22. *Given an instance language Li, a lower-admissible concept language Lc, and a membership M, the boundary set S of the version space of every possible task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$ *is finite.*

5.5.2 Algorithms of the Basic Version-Space Operations

This section presents algorithms of the basic version-space operations that are based on the UBSS representation. The theorems for correctness of the algorithms are given in Appendix B for the class of lower-admissible languages. Therefore, all the algorithms presented in this section are given for lower-admissible languages.

Note that the algorithms of the operation *Retraction*, *Subset?*, and *Equal?* are introduced for the first time. This is confirmed by the fact that the basic work of Hirsh (1992b) on the unilateral boundary sets has considered these three operations without implementation details.

5.5.2.1 Algorithm of the Operation *Initialise*

The operation *Initialise* sets the version space VS of a target concept when no training instances are available. The algorithm of the operation for the UBSS representation is proposed for the class of lower-admissible concept languages since the representation is a correct representation for this class.

The algorithm of the operation is presented in figure 5.7. Given a concept language Lc it initialises the UBSS representation of the version space VS by setting the minimal boundary set S to the most minimal descriptions in Lc. Thus, by theorem 5.21 the resulting boundary set S and the training set $I^- = \emptyset$ represent the initial version space VS equal to the concept language Lc.

Algorithm *Initialise*

Input: a lower-admissible concept language Lc.
Output: UBSS:$\langle S, \emptyset \rangle$ of a version space VS.

$S = MIN(Lc)$
return $\langle S, \emptyset \rangle$.

Figure 5.7: The Algorithm of the Operation *Initialise*.

Algorithm *Update*

Input: i: a new training instance.
UBSS:$\langle S, I^- \rangle$ of a version space VS.
Output: UBSS:$\langle S', I^{-\prime} \rangle$ of a version space VS' s.t.:
if i is a positive instance $\mathfrak{p}$ then $VS' = \{c \in VS | M(c, \mathfrak{p})\}$;
if i is a negative instance $\mathfrak{n}$ then $VS' = \{c \in VS | \neg M(c, \mathfrak{n})\}$.

if instance i is a positive instance $\mathfrak{p}$ **then**
$S' = MIN(\{c \in Lc | (\exists s \in S)(s \leq c) \wedge (\forall n \in I^-) \neg M(c, n) \wedge M(c, \mathfrak{p})\})$
$I^{-\prime} = I^-$
if instance i is a negative instance $\mathfrak{n}$ **then**
$S' = \{s \in S | \neg M(s, \mathfrak{n})\}$
$I^{-\prime} = I^- \cup \{\mathfrak{n}\}$
return $\langle S', I^{-\prime} \rangle$.

Figure 5.8: The Algorithm of the Operation *Update*.

5.5.2.2 Algorithm of the Operation *Update*

The operation *Update* revises the version space of a target concept when a new instance is added to the training sets. The algorithm of the operation for UBSS has two procedures for handling positive and negative training instances, respectively. The procedures are based on theorems B.1 and B.2 (see Appendix B). The theorems are correct for the class of lower-admissible concept languages. Thus, the algorithm is correct for this class as well.

The algorithm is given in figure 5.8. When a new positive training instance $\mathfrak{p}$ is given, the algorithm forms the boundary set S' of the updated version space VS' from the boundary set S of the initial version space VS and the negative training

set I^-. The set S' is formed from those minimal generalisations of the elements of the set S that (1) cover the instance $\mathfrak{p}$, and (2) do not cover any instance in the set I^-. Thus, by theorem B.1 the resulting minimal boundary set S' and the negative training set $I^{-\prime}$ ($I^{-\prime} = I^-$) correctly represent the version space VS'.

When a new negative training instance $\mathfrak{n}$ is given, the algorithm forms the boundary set S' of the updated version space VS' from those elements of the boundary set S of the initial version space VS that do not cover the instance $\mathfrak{n}$. The negative training set I^- is updated to a new one $I^{-\prime}$ such that $I^{-\prime} = I^- \cup \{\mathfrak{n}\}$. Thus, by theorem B.2 the resulting minimal boundary set S' and the negative training set $I^{-\prime}$ correctly represent the version space VS'.

5.5.2.3 Algorithm of the Operation *Retraction*

The operation *Retraction* revises the version space of a target concept when instances are removed from the training sets. The algorithm of the operation for UBSS has two procedures for handling positive and negative training instances, respectively. The procedures are based on theorems B.3 and B.4 (see Appendix B). The theorems are correct for the class of lower-admissible concept languages. Thus, the algorithm is correct for this class as well.

The algorithm is given in figure 5.9. When a positive training instance $\mathfrak{p}$ is retracted, the algorithm forms the boundary set S' of the new revised version space VS' from the boundary set S of the initial version space VS, the concept language Lc used, and the training sets I^+ and I^-. The set S' is formed equal to the minimal elements of the union of the minimal boundary set S and the minimal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$. Thus, by theorem B.3 the resulting minimal boundary set S' and the negative training set $I^{-\prime}$ ($I^{-\prime} = I^-$) correctly represent version space VS'.

When a negative training instance $\mathfrak{n}$ is retracted the algorithm forms the boundary set S' of the new revised version space VS' from the boundary set S of the initial version space VS, the concept language Lc used, and the training sets I^+ and I^-. The set S' is formed equal to the minimal elements of the union of the minimal boundary set S and the minimal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\}$. The set I^- is updated to a new one $I^{-\prime}$ such that $I^{-\prime} = I^- - \{\mathfrak{n}\}$. Thus, by theorem B.4 the resulting minimal boundary set S' and the negative training set $I^{-\prime}$ correctly represent version space VS'.

As with the case of the boundary sets the main problem of the algorithm of the operation *Retraction* is how to form the minimal boundary sets of the version spaces $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ and $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{n}\}, I^- \cup \{\mathfrak{n}\}\rangle)\}$. The general solution is to form these boundary sets by consequently applying the operation *Update*.

Algorithm *Retraction*
Input: i: a training instance to be retracted.
UBSS:$\langle S, I^- \rangle$ of a version space VS.
Lc: a lower-admissible concept language.
I^+: a set of positive training instances.
I^-: a set of negative training instances.
Output:
UBSS:$\langle S', I^{-\prime} \rangle$ of $VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\} \rangle)\}$
if i is a positive instance $\mathfrak{p}$.
UBSS:$\langle S', I^{-\prime} \rangle$ of $VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\} \rangle)\}$
if i is a negative instance $\mathfrak{n}$.

if instance i is a positive instance $\mathfrak{p}$ **then**
$S' = MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\} \rangle)\}))$
$I^{-\prime} = I^-$
if instance i is a negative instance $\mathfrak{n}$ **then**
$S' = MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\} \rangle)\}))$
$I^{-\prime} = I^- - \{\mathfrak{n}\}$
return $\langle S', I^{-\prime} \rangle$.

Figure 5.9: The Algorithm of the Operation *Retraction*.

5.5.2.4 Algorithm of the Operation *Collapsed?*

The operation *Collapsed?* determines whether a version space is empty. The algorithm of the operation is based on theorem B.5 (see Appendix B). Since the theorem is correct for the class of lower-admissible concept languages, the algorithm is correct for this class as well.

The algorithm checks a version space VS for collapse by checking the minimal boundary set S. If the set is not empty, then by theorem B.5 the version space VS is not empty and the algorithm returns false. Otherwise, it returns true; i.e., the version space VS is empty.

5.5.2.5 Algorithm of the Operation *Converged?*

The operation *Converged?* determines whether a version space has only one element[5]. The algorithm of the operation is based on theorem B.6 (see Appendix B). Since the theorem is correct for the class of lower-admissible concept languages, the algorithm is correct for this class as well.

[5] Note that the algorithm is given under assumption that the function $\mathcal{R}_{\mathfrak{C}}$ is injective. By lemma 3.13 this means that equivalent descriptions are equal.

Algorithm *Converged?*

Input: UBSS:$\langle S, I^- \rangle$ of a version space VS.
Output: true if $|VS| = 1$.
false if $|VS| \neq 1$.

if *Collapsed?*$(\langle S, I^- \rangle)$ **then**
return false
if $|S| = 1$ **then**
if $(\forall c \in MIN(\{c \in Lc | (\exists s \in S)(s < c)\}))(\exists n \in I^-)M(c, n)$ **then**
return true
return false.

Figure 5.10: The Algorithm of the Operation *Converged?*.

The algorithm of the operation *Converged?* is given in figure 5.10. To test a version space VS for convergence it executes two steps. In the first step the algorithm checks whether VS is empty using the algorithm of the operation *Collapsed?*. If so, then VS is not converged and by definition 3.12 the algorithm returns false. Otherwise, the algorithm executes the second step. It checks the size of the minimal boundary set S of VS. If the size is equal to one, then the algorithm generates all the minimal generalisations of the only element s of the set S. If each minimal generalisation covers at least one negative instance, then by theorem B.6 the version space VS has only one element s and the algorithm returns true. Otherwise, it returns false: the version space VS has more than one element.

5.5.2.6 Algorithm of the Operation *Classify*

The operation *Classify* determines the classification of instances using the rule of the unanimous vote of descriptions in version spaces. The algorithm of the operation is based on theorem B.7 (see Appendix B) in the case of positive classification. In the case of negative classification the algorithm is based on the algorithms of the operations *Update* and *Collapsed?* (see corollary 3.16 in subsection 3.2.3.1). Since the theorem and the algorithms are correct for the class of lower-admissible concept languages, the algorithm of the operation *Classify* is correct for this class as well.

The algorithm of the operation *Classify* is given in figure 5.11. When an instance has to be classified with a version space VS, then the algorithm checks whether VS is empty. If so, then according to definition 3.14 the algorithm returns "?". Otherwise, it checks whether the instance is covered by all the descriptions of the minimal boundary set S. If so, then by theorem B.7 all the descriptions of the version space

Algorithm *Classify*

Input: i: an instance to be classified.
UBSS:$\langle S, I^- \rangle$ of a version space VS.
Output: "+" if $(VS \neq \emptyset) \wedge (\forall c \in VS) M(c, i)$.
"−" if $(VS \neq \emptyset) \wedge (\forall c \in VS) \neg M(c, i)$.
"?" otherwise.

if *Collapsed?*$(\langle S, I^- \rangle)$ **then**
  return "?"
if $(\forall s \in S) M(s, i)$ **then**
  return "+"
label i as positive
$\langle S', I^{-\prime} \rangle = Update(i, \langle S, I^- \rangle)$
if *Collapsed?*$(\langle S', I^{-\prime} \rangle)$ **then**
  return "−"
return "?".

Figure 5.11: The Algorithm of the Operation *Classify*.

VS cover the instance and according to definition 3.2.3 "+" is returned. Otherwise, the algorithm updates the version space VS with the instance i labelled as positive. If VS collapses then by corollary 3.16 all the descriptions of the version space VS do not cover the instance and according to definition 3.2.3 "−" is returned. If positive and negative classification of the instance cannot be determined the algorithm returns "?" (according to definition 3.2.3).

5.5.2.7 Algorithm of the Operation *Member?*

The operation *Member?* determines whether a concept description belongs to a version space. The algorithm of the operation is based on theorem 5.21. Since the theorem is correct for the class of lower-admissible concept languages, the algorithm is correct for this class as well.

When a concept description c has to be checked for membership of a version space VS the algorithm tests:

(1) the existence of at least one element of the minimal boundary set S that is more specific than c; and

(2) the consistency of c with respect to the set I^- of negative instances; i.e., whether c does not cover any negative instance.

Algorithm *Intersection*

Input: UBSS:$\langle S_1, I_1^- \rangle$ of a version space VS_1.
UBSS:$\langle S_2, I_2^- \rangle$ of a version space VS_2.
Output: UBSS:$\langle S_{12}, I_{12}^- \rangle$ of a version space $VS_{12} = VS_1 \cap VS_2$.

$S_{12} = MIN(\{c \in Lc | (\exists s_1 \in S_1)(\exists s_2 \in S_2)((s_1 \leq c) \wedge (s_2 \leq c))\})$
$S_{12} = \{s \in S_{12} | (\forall n \in I_1^- \cup I_2^-) \neg M(s, n)\}$
$I_{12} = I_1^- \cup I_2^-$
return $\langle S_{12}, I_{12}^- \rangle$.

Figure 5.12: The Algorithm of the Operation *Intersection*.

If the conditions (1) and (2) hold, then by theorem 5.21 the description c belongs to the version space VS and the algorithm returns true. Otherwise, it returns false: c does not belong to the version space VS.

5.5.2.8 Algorithm of the Operation *Intersection*

The operation *Intersection* computes a version space VS_{12} that is the intersection of two other version spaces VS_1 and VS_2. The algorithm of the operation for UBSS is based on theorem B.8 (see Appendix B). Since the theorem is correct for the class of lower-admissible concept languages, the algorithm is correct for this class as well.

The algorithm of the operation *Intersection* is given in figure 5.12. Its input is the UBSS representations $\langle S_1, I_1^- \rangle$ and $\langle S_2, I_2^- \rangle$ of two version spaces VS_1 and VS_2. The algorithm computes the UBSS representation $\langle S_{12}, I_{12}^- \rangle$ of the version space VS_{12}, that is the intersection of VS_1 and VS_2 as follows. The set S_{12} is formed in two steps. In the first step the set is initialised equal to the set of minimal generalisations of the elements of the minimal boundary sets S_1 and S_2. In the second step the set S_{12} is pruned: all the elements that do cover any negative instance of the set $I_1^- \cup I_2^-$ are removed. The set I_{12}^- is formed equal to the union $I_1^- \cup I_2^-$. Thus, by theorem B.8 the resulting minimal boundary set S_{12} and the negative training set I_{12}^- correctly represent the version space VS_{12}.

5.5.2.9 Algorithm of the Operation *Subset?*

The operation *Subset?* determines whether a version space VS_1 is a subset of another version space VS_2. The algorithm of the operation is based on theorem B.9 (see Appendix B). Since the theorem B.9 is correct for the class of lower-admissible concept languages, the algorithm is correct for this class as well.

Algorithm *Subset?*

Input: UBSS:$\langle S_1, I_1^- \rangle$ of a version space VS_1.
UBSS:$\langle S_2, I_2^- \rangle$ of a version space VS_2.
Output: true if $(VS_1 \subseteq VS_2)$.
false if $\neg(VS_1 \subseteq VS_2)$.

if $(\exists s_1 \in S_1)(\forall s_2 \in S_2)\neg(s_2 \leq s_1)$ **then**
return false
for $n \in I_2^-$ **do**
if *Classify*$(n, \langle S_1, I_1^- \rangle) \neq$ "$-$" **then**
return false
return true.

Figure 5.13: The Algorithm of the Operation *Subset?*.

The algorithm is given in figure 5.13. Its input is the UBSS representations $\langle S_1, I_1^- \rangle$ and $\langle S_2, I_2^- \rangle$ of two version spaces VS_1 and VS_2. To test whether VS_1 is a subset of VS_2 the algorithm executes two steps. In the first step it checks whether there exists at least one element s_1 of the minimal boundary set S_1 that is not more general than any element s_2 of the minimal boundary set S_2 in the partially-ordered structure of the concept language used. If so, by theorem B.9 VS_1 is not a subset of VS_2 and the algorithm returns false. Otherwise, the algorithm executes the second step: it classifies each training instance n of the set I_2^- with the version space VS_1. The instance classification is realised with the algorithm of the operation *Classify* applied on the UBSS representation of the version space VS_1. Thus, if the classification of at least one instance n in the set I_2^- is not negative by theorem B.9 VS_1 is not a subset of VS_2 and the algorithm returns false. Otherwise it returns true: VS_1 is a subset of VS_2.

5.5.2.10 Algorithm of the Operation *Equal?*

The operation *Equal?* determines whether a version space VS_1 is equal to another version space VS_2. The algorithm of the operation is based on theorem B.10 (see Appendix B). Since the theorem B.10 is correct for the class of lower-admissible concept languages the algorithm is correct for this class as well.

The algorithm is given in figure 5.14. Its input is the UBSS representations $\langle S_1, I_1^- \rangle$ and $\langle S_2, I_2^- \rangle$ of two version spaces VS_1 and VS_2. To test whether VS_1 is a equal to VS_2 the algorithm executes three steps. In the first step it tests whether the sets S_1 and S_2 are different. If so, then by theorem B.10 VS_1 is not equal to VS_2,

Algorithm *Equal?*

Input: UBSS:$\langle S_1, I_1^- \rangle$ of a version space VS_1.
UBSS:$\langle S_2, I_2^- \rangle$ of a version space VS_2.
Output: true if $(VS_1 = VS_2)$
false if $(VS_1 \neq VS_2)$

if $S_1 \neq S_2$ **then**
return false
for $n \in I_1^-$ **do**
if *Classify*$(n, \langle S_2, I_2^- \rangle) \neq$ "$-$" **then**
return false
for $n \in I_2^-$ **do**
if *Classify*$(n, \langle S_1, I_1^- \rangle) \neq$ "$-$" **then**
return false
return true.

Figure 5.14: The Algorithm of the Operation *Equal?*.

and the algorithm returns false. Otherwise, the algorithm executes the second step: it classifies each training instance n of the set I_1^- with the version space VS_2. The instance classification is realised with the algorithm of the operation *Classify* applied over the UBSS representation of the version space VS_2. Thus, if the classification of at least one instance n in the set I_1^- is not negative, by theorem B.9 VS_1 is not equal to VS_2, and the algorithm returns false. Otherwise, the algorithm executes the third step: it classifies each training instance n of the set I_2^- with the version space VS_1. The instance classification is realised with the algorithm of the operation *Classify* applied over the UBSS representation of the version space VS_1. Thus, if the classification of at least one instance n in the set I_2^- is not negative, by theorem B.9 VS_1 is not equal to VS_2, and the algorithm returns false. Otherwise it returns true: VS_1 is equal to VS_2.

5.5.3 Worst-Case Complexity Analysis

This subsection presents a worst-case complexity analysis of the UBSS algorithms of the basic version-space operations. The analysis is made under the worst-case assumption (1) from subsection 5.4.3 in terms of:

- the number P of positive instances;
- the number N of negative instances;

- the largest size Σ of the set $MIN(Lc)$;
- the worst-case time complexity $t^{\downarrow}$ for generating the set $MIN(Lc)$;
- the maximal number $\check{g}$ of minimal generalisations of a concept description.
- the worst-case time complexity t_g to determine all the minimal generalisations of a concept description;
- the worst-case time complexity t_c to test whether one concept description covers another concept description or an instance;

5.5.3.1 Worst-Case Space-Complexity Analysis of the Representation

In the worst case the minimal boundary set S has $\Sigma\check{g}^P$ elements[6]. Thus, since the set I^- of negative instances has to be stored, the size of the UBSS representation is equal to $O(\Sigma\check{g}^P + N)$ in the worst-case.

5.5.3.2 Worst-Case Time-Complexity Analysis of the Algorithms

This subsection presents a worst-case time complexity analysis of the algorithms of the basic version-space operations based on the UBSS representation.

The Algorithm of the Operation *Initialise*. Since the algorithm initialises only the minimal boundary set S, its worst-case time complexity is equal to $O(t^{\downarrow})$ (see figure 5.7).

The Algorithm of the Operation *Update*. The algorithm consists of two procedures for handling positive and negative training instances, respectively. Hence, the time complexity analysis is realised for each of the procedures separately.

The worst-case time complexity of the procedure for processing a positive training instance is equal to $O(|S|t_g + |S|N\check{g}t_c + |S|\check{g}t_c + (|S|\check{g})^2t_c)$ (see figure 5.8). The term $O(|S|t_g)$ arises because each element of the minimal boundary set S has to be minimally generalised to cover the instance. The term $O(|S|N\check{g}t_c)$ arises because $|S|\check{g}$ minimal generalisations are checked whether they cover the instances of the negative training set. The term $O(|S|\check{g}t_c)$ arises because $|S|\check{g}$ minimal generalisations are checked whether they cover the positive training instance. The term $O((|S|\check{g})^2t_c)$ arises because each two minimal generalisations are compared so that the non-minimal elements are removed. Thus, the overall worst-case time complexity is equal to $O(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c)$.

The worst-case time complexity of the procedure for processing a negative training instance is $O(|S|t_c)$. This is due to the fact that the elements of the minimal boundary set S are checked whether they cover the negative instance.

[6] This can be proven by induction on the size of the number P.

The Algorithm of the Operation *Retraction*. The algorithm consists of two procedures for handling positive and negative training instances, respectively. Thus, the time complexity analysis is realised for each of the procedures separately.

The procedure for handling one positive instance generates the minimal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ (see figure 5.9). The worst-case size of this minimal boundary set is $O(\Sigma\check{g}^{P-1})$. We substitute this size in the worst-case time-complexity expressions of the algorithm of the operation *Update*. Thus, after $P-1$ positive and $N+1$ negative training instances:

- the worst-case time complexity of the procedure of the operation *Update* given a positive instance is $O(\Sigma\check{g}^{P-1}t_g + \Sigma\check{g}^{P-1}(N+1)\check{g}t_c + (\Sigma\check{g}^{P-1}\check{g})^2 t_c)$;
- the worst-case time complexity of the procedure of the operation *Update* given a negative instance is $O(\Sigma\check{g}^{P-1}t_c)$.

To generate the minimal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ we need to apply the algorithm of the operation *Initialise*, $P-1$ times the procedure of the operation *Update* given a positive instance, and $N+1$ times the procedure of the operation *Update* given a negative instance. Thus, the worst-case time complexity for generating these sets is $O(t^{\downarrow} + (\Sigma\check{g}^{P-1}t_g + \Sigma\check{g}^{P-1}(N+1)\check{g}t_c + (\Sigma\check{g}^{P-1}\check{g})^2 t_c) + (\Sigma\check{g}^{P-1}t_c))$[7]. After a simplification it is $O(t^{\downarrow} + \Sigma\check{g}^{P-1}t_g + \Sigma\check{g}^{P}(N+1)t_c + \Sigma^2\check{g}^{2P}t_c)$. The elements of the minimal boundary set of the version space $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ are added to the minimal boundary set S of the resulting version space. Therefore, each two elements of the set S are compared so that the non-minimal elements are removed. The worst-case time complexity of this calculation is $O(|S|\Sigma\check{g}^{P-1}t_c)$. Thus, worst-case time complexity of the procedure for handling one positive instance is $O(t^{\downarrow} + \Sigma\check{g}^{P-1}t_g + \Sigma\check{g}^{P}(N+1)t_c + \Sigma^2\check{g}^{2P}t_c + |S|\Sigma\check{g}^{P-1}t_c)$.

The worst-case time complexity of the procedure for handling one negative instance is derived by analogy. It is $O(t^{\downarrow} + \Sigma\check{g}^{P+1}t_g + \Sigma\check{g}^{P+2}(N-1)t_c + \Sigma^2\check{g}^{2(P+2)}t_c + |S|\Sigma\check{g}^{P+1}t_c)$.

The Algorithm of the Operation *Collapsed?*. The worst-case time complexity of the algorithm is $O(1)$. This is due to the fact that the algorithm checks whether the minimal boundary set S is empty.

The Algorithm of the Operation *Converged?*. The worst-case time complexity of the algorithm is $O(1+1+t_g+N\check{g}t_c)$ (see figure 5.10). The first term $O(1)$ arises because it is the worst-time complexity of the algorithm of the operation *Collapsed?*. The second term $O(1)$ arises because it is the time complexity to check whether the size of the minimal boundary set S is equal to one. The term $O(t_g)$ arises because it is the worst-time complexity for generating all the minimal generalisations of the

[7] Note that only the worst-case time complexities of the procedures of the operation *Update* after $P-1$ and $N+1$ have an importance for forming this complexity.

only element of the set S. The term $O(N\check{g}t_c)$ arises because all the minimal generalisations are checked whether they cover the instances from the negative training set. Thus, the worst-case time complexity of the algorithm is $O(t_g + N\check{g}t_c)$.

The Algorithm of the Operation *Classify*. The worst-case time complexity of positive classification is equal to $O(|S|t_c)$ since the elements of the minimal boundary set S are checked whether they cover the instance to be classified (see figure 5.11).

The worst-case time complexity of the negative classification is equal to the time complexity $O(|S|t_g+|S|N\check{g}t_c+(|S|\check{g})^2t_c+1)$. The term $O(|S|t_g+|S|N\check{g}t_c+(|S|\check{g})^2t_c)$ arises because it is the time complexity of the algorithm of the operation *Update* given a positive training instance. The term $O(1)$ arises because it is the time complexity of the operation *Collapsed?*. Thus, the worst-case time complexity of the negative classification is $O(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c)$.

The worst-case time complexity of the algorithm of the operation *Classify* is equal to the sum of the worst-case time complexity of the algorithm of the operation *Collapsed?*, the worst-case time complexity of the positive classification, and the worst-case time complexity of the negative classification. Thus, it is equal to $O(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c)$.

The Algorithm of the Operation *Member?*. The worst-case time complexity of the algorithm is $O(|S|t_c+Nt_c)$. The term $O(|S|t_c)$ arises because the elements of the minimal boundary set S are checked to be more specific than a concept description which membership has to be determined. The term $O(Nt_c)$ arises because the concept description is checked whether it covers negative instances.

The Algorithm of the Operation *Intersection*. The worst-case time complexity of the algorithm is equal to the worst-case time complexity for generating the minimal boundary set S_{12} (see figure 5.12). The time complexity for generating the set S_{12} is $O(2|S|t_g + (|S|\check{g})^2t_c + (|S|\check{g})^2t_c + |S|Nt_c)$, where S is the largest set among the sets S_1 and S_2, and N is the greatest number among the numbers N_1 and N_2. The term $O(2|S|t_g)$ arises because the elements of both sets S_1 and S_2 have to be minimally generalised. The first term $O((|S|\check{g})^2t_c)$ arises because $|S|\check{g}$ minimal generalisations of the set S_1 are compared with $|S|\check{g}$ minimal generalisations of the set S_2 in order to determine common minimal generalisations. The second term $O((|S|\check{g})^2t_c)$ arises because each two minimal common generalisations are compared so that the non-minimal elements are removed. The term $O(|S|Nt_c)$ arises because each minimal generalisation of the set S_{12} is checked whether it covers the instances of the negative training sets. Thus, the overall complexity for generating the set S_{12} is $O(|S|t_g + (|S|\check{g})^2t_c + |S|Nt_c)$.

The Algorithm of the Operation *Subset?*. The worst-case time complexity of the algorithm is equal to $O(|S|^2t_c + N(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c))$, where S is the largest set among the sets S_1 and S_2, and N is the greatest number among the numbers N_1 and N_2 (see figure 5.13). The term $O(|S|^2t_c)$ arises because each element of the minimal boundary set S_1 is compared with all the elements of the minimal boundary set S_2 at the worst case. The term $O(N(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c))$

arises because N instances of the set I_2^- are classified with the algorithm of the operation *Classify* of which the worst-case time complexity of negative classification is $O(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c)$. Thus, the overall worst-case time complexity is $O(N(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c))$.

The Algorithm of the Operation *Equal?*. The worst-case time complexity of the algorithm is equal to $O(|S|^2t_c + 2N(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c))$, where S is the largest set among the sets S_1 and S_2, and N is the greatest number among the numbers N_1 and N_2 (see figure 5.14). The term $O(|S|^2t_c)$ arises because each element of the minimal boundary set S_1 is compared with all the elements of the minimal boundary set S_2 at the worst case. The term $O(2N(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c))$ arises because N instances of each of the sets I_1^- and I_2^- are classified with the algorithm of the operation *Classify* of which the worst-case time complexity of negative classification is $O(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c)$. Thus, the overall worst-case time complexity is $O(N(|S|t_g + |S|N\check{g}t_c + (|S|\check{g})^2t_c))$.

5.5.4 Adequacy of the Representation

In this subsection we investigate the tractability and the epistemological and heuristical adequacy of the unilateral-boundary-set representation for the basic version-space operations.

5.5.4.1 Epistemological Adequacy for the Basic Version-Space Operations

Subsection 5.5.2 has presented the UBBS algorithms of all the basic version-space operations. The correctness of the algorithms has been proven for lower-admissible languages (see (Hirsh, 1992b) and Appendix B). Thus, the UBSS representation is epistemologically adequate with respect to all the basic version-space operations for the class of lower-admissible concept languages.

5.5.4.2 Heuristical Adequacy for the Basic Version-Space Operations

We investigate the heuristical adequacy of the UBSS representation for the class of lower-admissible concept languages. We use the results of the worst-case complexity analysis from subsection 5.5.3. They show that the time complexities of the algorithms of the basic version-space operations *Initialise, Update, Collapsed?, Converged?, Classify, Member?, Intersection, Subset?* and *Equal?* are polynomial in the sizes of the input $|S|$ and N, the complexities $t^{\downarrow}$, t_c, t_g, and the maximal number $\check{g}$. Thus, the algorithms of these operations are tractable; i.e., the UBSS representation is heuristically adequate for these operations for the class of lower-admissible concept languages, if the complexities $t^{\downarrow}$, t_c, t_g, and the maximal number $\check{g}$ are polynomial in relevant properties of these concept languages.

The algorithm of the basic version-space operation *Retraction* requires additional attention. Its time complexity is polynomial in the size of the input $|S|$ and N, the complexities $t^{\downarrow}$, t_c, t_g, but it is exponential in the number P of positive training instances with a base the maximal number $\breve{g}$. Thus, in general the algorithm of the operation *Retraction* is intractable. Nevertheless, for a particular case if the complexities $t^{\downarrow}$, t_c, t_g are polynomial in relevant properties of the concept languages used, and the maximal number $\breve{g}$ is equal to one, the algorithm is tractable.

5.5.4.3 Tractability

We investigate the tractability of the UBSS representation for the class of lower-admissible concept languages. According to the terminology in section 5.1 the UBBS are tractable for a concept language when they are heuristically adequate for the operations *Initialise* and *Update*, and their size is polynomial in the number of training instances and relevant language properties. In this context we have established that:

- the UBSS are heuristically adequate for the operation *Initialise* if the generation complexity $t^{\downarrow}$ is polynomial in the relevant properties of the concept languages used;
- the UBSS are heuristically adequate for the operation *Update* if the complexities t_c and t_g, and the maximal number $\breve{g}$ are polynomial in relevant properties of the concept languages used;
- the size of the minimal boundary set is exponential in the numbers P of positive training instances with a base the maximal number $\breve{g}$.

Thus, in general the UBSS representation is intractable for the class of lower-admissible concept languages. Nevertheless, for a particular case if (1) the complexities $t^{\downarrow}$, t_c, t_g are polynomial in relevant properties of the concept languages used, and (2) the maximal number $\breve{g}$ is equal to one, the representation is tractable.

5.6 Chapter Conclusion

This chapter has considered the problem of implementing the version-space abstract data type. In this context we have surveyed three well-known version-space representations: list representation, boundary sets, and unilateral boundary sets. They have been considered along with their algorithms of the basic version-space operations. Since our intention has been to systematise the representations we have obtained new results for the boundary sets and the unilateral boundary sets. More precisely, we have proven that:

(1) both representations have algorithms for the operations *Retraction* and *Subset?* for concept languages for which they were designed. Hence, the representations are epistemologically adequate with respect to the operations *Retraction* and *Subset?* for these languages. This extends the previous work of Mitchell (1978, 1997) and Hirsh (1992b) where either the operations were not defined or their implementation details were not given.

(2) the boundary sets are epistemologically adequate for the operations *Converged?*, *Collapsed?* and *Classify* for the class of admissible concept languages. This sounds trivially since the algorithms of the operations were already introduced in (Mitchell, 1978), but their correctness was never proven.

(3) the unilateral boundary sets have an algorithm for the operation *Equal?* for lower-admissible and upper-admissible concept languages. Hence, they are epistemologically adequate for the operation for these classes of languages. This completes the previous basic work of Hirsh (1992b) on the unilateral boundary sets where the operation was considered without implementation details.

The results (1), (2), and (3) are the main contribution of this chapter. They have extended the epistemological adequacy of the boundary sets and the unilateral boundary sets for the whole set of the basic version-space operations. Therefore, we have derived the conditions for the tractability and the heuristical adequacy of the representations for the same set of operations. Hence, we can conclude that *this chapter has completed the research on the tractability and the epistemological and heuristical adequacy of the boundary sets and the unilateral boundary sets for the basic version-space operations.*

In addition, *the chapter has shown that the presented version-space representations are computationally inefficient for the operation Retraction.* The algorithm of the operation for the list representation is natural, but it is tractable only for small concept languages (see section 5.2). This constraint of applicability is overcome by the boundary sets, but (a) they are intractable for most concept languages, and (b) the complexity of their algorithm of the operation *Retraction* is at least the number of training instances times the complexity of the algorithm of the operation *Update*. Drawback (a), the computational problem of boundary sets, can be partially avoided by the unilateral boundary sets, but drawback (b), the retraction problem, cannot. Hence, we conclude that we need a version-space representation that simultaneously solves the computational problem of boundary sets and the retraction problem; i.e., we need a version space representation that simultaneously solves the second and third part of our problem statement. We consider the development of a such version-space representation in the next chapter.

Chapter 6

Instance-Based Boundary Sets

This chapter answers the second and third part of our problem statement. It proposes a new alternative version-space representation that simultaneously solves the computational and retraction problems of the boundary sets. The representation is derived from an analysis given in section 6.1. The key idea is to consider the version space of a target concept as an intersection of version spaces with respect to positive and negative training instances[1]. We show that taken together the minimal boundary sets of version spaces with respect to positive instances and the maximal boundary sets of version spaces with respect to negative instances delimit the version space. Therefore, an ordered pair of families of these minimal and maximal boundary sets is proposed as a version space representation in section 6.2. Each element of the representation is a boundary set and it is associated with a particular training instance. Thus, the representation is called *instance-based boundary sets* (Smirnov and Braspenning, 1998a, 1998b).

In section 6.2 we prove that the instance-based boundary sets represent correctly version spaces for the class of admissible concept languages. Hence, all elements of version spaces are considered without explicit enumeration.

In order to study epistemological adequacy of the instance-based boundary sets for the basic version-space operations we introduce classes of intersection-preserving and union-preserving languages. The classes are defined in section 6.3 as subclasses of admissible concept languages. This allows us in section 6.4 to present the algorithms of the operations *Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset?* and *Equal?*, that are based on the instance-based boundary sets. The algorithms are proven to be correct for either the class of admissible languages or the classes of intersection-preserving and union-preserving languages. Their worst-case complexity analysis is given in section 6.5.

[1] The version spaces with respect to negative instances have been determined in definition 4.13. The version spaces with respect to positive instances can be defined by duality.

The adequacy of the instance-based boundary sets is studied in section 6.6. We show that the representation is epistemologically adequate for the basic version-space operations above for the classes of intersection-preserving and union-preserving languages in subsection 6.6.1. Therefore, in subsections 6.6.2 and 6.6.3 we derive the conditions when the instance-based boundary sets are heuristically adequate for the operations and are tractable. The conditions are specialised for 1-CNF languages with Boolean attributes in subsection 6.6.4. They are experimentally verified in section 6.7.

In section 6.8 a comparison with relevant work is given. Subsection 6.9 contains chapter conclusions.

6.1 Class of Instance-Based Boundary Sets

This section introduces a class of instance-based boundary-set representations of version spaces. Introducing the class starts with two dual lemmas 6.1 and 6.2. The lemmas state that the version space VS of a target concept can be regarded as an intersection of two types of version spaces, namely version spaces with respect to positive instances and version spaces with respect to negative instances (see definition 4.13).

Lemma 6.1. *Consider the version space VS of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. If $I^+ \neq \emptyset$ then:*

$$VS = \bigcap_{p \in I^+} VS(p)$$

where $VS(p) = \{c \in Lc | cons(c, \langle \{p\}, I^- \rangle)\}$.

Proof. The lemma is a direct corollary of theorem 3.20. □

Lemma 6.2. *Consider the version space VS of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. If $I^- \neq \emptyset$ then:*

$$VS = \bigcap_{n \in I^-} VS(n)$$

where $VS(n) = \{c \in Lc | cons(c, \langle I^+, \{n\} \rangle)\}$.

Proof. The lemma is a direct corollary of theorem 3.20. □

According to definition 4.13 a version space $VS(n)$ with respect to a negative instance is a set of all the descriptions in the concept language that are consistent with the set of positive instances and that instance. By duality, a version space $VS(p)$ with respect to a positive instance is a set of all the descriptions in the concept

language that are consistent with that instance and the set of negative instances. Thus, the version spaces $VS(p)$ and $VS(n)$ are supersets of the version space VS to be learned (see corollaries 6.3 and 6.4 given below).

Corollary 6.3. *Consider the version space VS of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. If $I^+ \neq \emptyset$ then:*

$$(\forall p \in I^+)(VS(p) \supseteq VS).$$

Proof. The lemma is a direct corollary of theorem 3.22. □

Corollary 6.4. *Consider the version space VS of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. If $I^- \neq \emptyset$ then:*

$$(\forall n \in I^-)(VS(n) \supseteq VS).$$

Proof. The lemma is a direct corollary of theorem 3.22. □

We use the corollaries in order to consider four properties of the boundary sets of version spaces $VS(p)$ and $VS(n)$ when they are defined in admissible concept languages:

(1) By corollary 6.3 and theorem 5.18 the elements of the minimal boundary sets $S(p)$ of the version spaces $VS(p)$ are more specific than the elements of the minimal boundary set S of the version space VS to be learned. Hence, we conclude that:

   (c1) if a description c in the concept language Lc is more general than at least one element of the minimal boundary set $S(p)$ for all $p \in I^+$, then c is consistent with the set I^+ of positive instances.

(2) By corollary A.4 (see Appendix A) the elements of the minimal boundary sets $S(n)$ of the version spaces $VS(n)$ are supersets of the minimal boundary set S of the version space VS to be learned. Hence, we conclude that:

   (c2) if a description c in the concept language Lc is more general than at least one element of some minimal boundary set $S(n)$, then c is consistent with the set I^+ of positive instances.

(3) By corollary A.3 (see Appendix A) the elements of the maximal boundary sets $G(p)$ of the version spaces $VS(p)$ are supersets of the maximal boundary set G of the version space VS to be learned. Hence, we conclude that:

(c3) if a description c in the concept language Lc is more specific than at least one element of some maximal boundary set $G(p)$, then c is consistent with the set I^- of negative instances.

(4) By corollary 6.4 and theorem 5.18 the elements of the maximal boundary sets $G(n)$ of the version spaces $VS(n)$ are more general than the minimal boundary set S of the version space VS to be learned. Hence, we conclude that:

(c4) if a description c in the concept language Lc is more specific than at least one element of the maximal boundary sets $G(n)$ for all $n \in I^-$, then c is consistent with the set I^- of negative instances.

Thus, if a description c satisfies simultaneously (c1) and (c3), (c1) and (c4), (c2) and (c3), or (c2) and (c4), then it can be concluded that c belongs to the version space VS. Therefore, we can identify four types of version-space representations:

- $\langle \{S(p)\}_{p \in I^+}, G(p_G) \rangle$ for some $p_G \in I^+$;
- $\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$;
- $\langle S(n_S), G(p_G) \rangle$ for some $n_S \in I^-$ and $p_G \in I^+$;
- $\langle S(n_S), \{G(n)\}_{n \in I^-} \rangle$ for some $n_S \in I^-$.

Note that each element of these representations is a boundary set and is based on a particular training instance. Hence, *the representations form a class of the instance-based boundary-set representations.*

To determine which representation is more efficient a complexity analysis is carried out. Thus, four additional properties are derived:

(5) By theorem A.1 (see Appendix A) the minimal boundary set $S(p)$ of a version space $VS(p)$ does not grow in the number of positive training instances. By theorem A.2 (see Appendix A) it can become smaller in the number of negative training instances.

(6) The minimal boundary set $S(n)$ of a version space $VS(n)$ can exponentially grow in the number of positive training instances (see section 5.4.3 and (Hirsh *et al.*, 1997)).

(7) The maximal boundary set $G(p)$ of a version space $VS(p)$ can exponentially grow in the number of negative training instances (see section 5.4.3 and (Haussler, 1988)).

(8) By theorem A.2 (see Appendix A) the maximal boundary set $G(n)$ of a version space $VS(n)$ does not grow in the number of negative training instances. By theorem A.1 (see Appendix A) it can become smaller in the number of positive training instances.

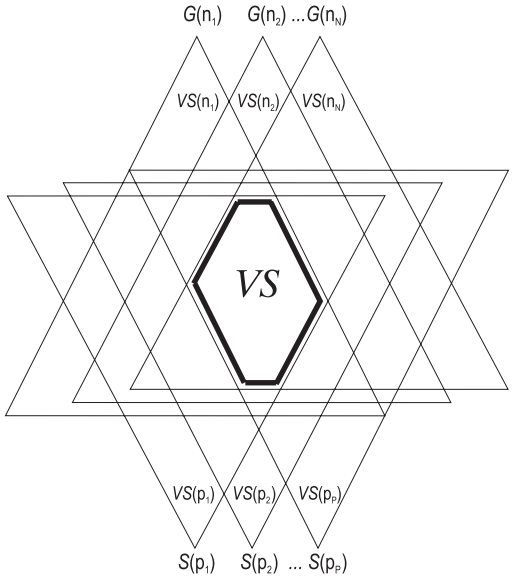

Figure 6.1: Version Space and Instance-Based Boundary Sets

From (5) and (8) on the one hand and (6) and (7) on the other it follows that the second representation $\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$ is the most computationally efficient.

6.2 Instance-Based Boundary Sets: Definition and Correctness

Since the representation $\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$ is most efficient it is chosen as the instance-based boundary-set representation of version spaces considered in this thesis. It is formally introduced in definition 6.5 and shown in figure 6.1.

Definition 6.5. (Instance-based Boundary Sets (IBBS)) *Consider the version space VS of a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle \rangle$. *If the concept language Lc is admissible, then the version space VS is represented by an ordered pair of indexed families of boundary sets* $\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$, *where:*

$$S(p) = MIN(VS(p)) \text{ for all } p \in I^+$$
$$G(n) = MAX(VS(n)) \text{ for all } n \in I^-.$$

To be precise, we repeat some of the characteristics of the IBBS from the previous section. The IBBS are "instance-based" since they contain the minimal and maximal boundary sets $S(p)$ and $G(n)$ of the version spaces $VS(p)$ and $VS(n)$ that correspond to particular training instances. The IBBS are "boundary sets" since each of their elements is a boundary set $S(p)$ or $G(n)$.

The indexed families $\{S(p)\}_{p\in I^+}$ and $\{G(n)\}_{n\in I^-}$ of boundary sets are called S and G-parts of the IBBS, respectively. In the previous section we have discovered that they have two dual properties (c1) and (c4). The properties are formalised in lemmas 6.6 and 6.7, respectively.

Lemma 6.6. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ *with version space VS represented by IBBS:*$\langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle$. *If the concept language Lc is admissible then:*

$$(\forall c \in Lc)((\forall p \in I^+)(\exists s \in S(p))(s \leq c) \rightarrow (\forall p \in I^+)M(c,p)).$$

Proof. We prove the first part of lemma. Let $c \in Lc$ be arbitrarily chosen such that $(\forall p \in I^+)(\exists s \in S(p))(s \leq c)$. Consider arbitrary $p \in I^+$ and its corresponding minimal boundary set $S(p)$ together with an element $s \in S(p)$ such that $s \leq c$. Since Lc is admissible, by lemma 5.5 $s \leq c$ and $M(s,p)$ imply $M(c,p)$. The last derivation holds for all $p \in I^+$. Thus, $(\forall p \in I^+)M(c,p)$. □

Lemma 6.7. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ *with version space VS represented by IBBS:*$\langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle$. *If the concept language Lc is admissible then:*

$$(\forall c \in Lc)((\forall n \in I^-)(\exists g \in G(n))(c \leq g) \rightarrow (\forall n \in I^-)\neg M(c,n)).$$

Proof. The proof is dual to that of lemma 6.6. □

We use lemmas 6.6 and 6.7 to prove that the IBBS correctly represent version spaces for the class of admissible concept languages. The proof is given in theorem 6.8.

Theorem 6.8. (Correctness of IBBS) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ *with version space VS represented by IBBS:*$\langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle$. *If the concept language Lc is admissible, then:*

$$(\forall c\in Lc)((c\in VS) \leftrightarrow ((\forall p\in I^+)(\exists s\in S(p))(s \leq c) \wedge (\forall n\in I^-)(\exists g\in G(n))(c \leq g))).$$

Proof. ($\rightarrow$) Consider an arbitrarily chosen $c \in VS$. By lemmas 6.1 and 6.2:

$$(\forall p \in I^+)(c \in VS(p))$$
$$(\forall n \in I^-)(c \in VS(n)).$$

But, Lc is admissible. Thus, by theorem 5.18:

$$(\forall p \in I^+)(\exists s \in S(p))(s \leq c)$$
$$(\forall n \in I^-)(\exists g \in G(n))(c \leq g),$$

and the first part of the theorem is proven.

($\leftarrow$) Consider an arbitrarily chosen $c \in Lc$ such that:

$$(\forall p \in I^+)(\exists s \in S(p))(s \leq c) \tag{6.1}$$
$$(\forall n \in I^-)(\exists g \in G(n))(c \leq g). \tag{6.2}$$

By lemma 6.6 formula (6.1) implies:

$$(\forall p \in I^+)M(c,p). \tag{6.3}$$

By lemma 6.7 formula (6.2) implies:

$$(\forall n \in I^-)\neg M(c,n). \tag{6.4}$$

According to definition 2.22 (6.3) and (6.4) are equivalent to:

$$cons(c, \langle I^+, I^- \rangle).$$

Thus, since $c \in Lc$ by definition of version spaces we conclude that $c \in VS$, and the second part of the theorem is proven. □

The theorem states that the IBBS correctly represent the version space VS to be learned for the class of admissible concept languages. The concept descriptions in VS are exactly those that are (1) more general than at least one element of each minimal boundary set $S(p)$, and (2) more specific than at least one element of each maximal boundary set $G(n)$. Thus, the IBBS preserve the main principal features of the boundary sets, namely all the elements of VS are considered, and all the elements of VS do not have to be explicitly enumerated.

By constraint 5.19 the minimal and maximal boundary sets of every version space are finite. This implies that the minimal and maximal boundary sets $S(p)$ and $G(n)$ are finite. By constraint 2.21 the training sets are finite. Thus, by definition 6.5 the IBBS are finite. This means that we can build algorithms that are based on the IBBS.

6.3 Intersection-Preserving and Union-Preserving Languages

Some of the algorithms of the basic version-space operations, based on the IBBS, assume that concept languages are either intersection-preserving or union-preserving. That is why in this section we introduce these two dual subclasses of admissible concept languages. We start with intersection-preserving languages in definition 6.9 below.

Definition 6.9. (Intersection-Preserving Languages (IPL)) *An admissible concept language Lc is an intersection preserving language if and only if for every nonempty subset* $C \subseteq Lc$*:*

(1) $glb(C) \in Lc$*; and*

(2) $(\forall i \in Li)((\forall c \in C)M(c,i) \leftrightarrow M(glb(C),i))$.

By the definition above every nonempty subset C in an IPL concept language Lc has a greatest lower bound $glb(C)$ such that if an instance i is covered by all the concept descriptions $c \in C$, then it is covered by $glb(C)$ as well. Thus, the extensional representation of a concept given by $glb(C)$ is the intersection of the extensional representations of the concepts given by the descriptions in C. This explains the name of the IPL languages.

The union-preserving languages are defined by duality. More formally:

Definition 6.10. (Union-Preserving Languages (UPL)) *An admissible concept language Lc is a union-preserving language if and only if for every nonempty subset* $C \subseteq Lc$*:*

(1) $lub(C) \in Lc$*; and*

(2) $(\forall i \in Li)((\exists c \in C)M(c,i) \leftrightarrow M(lub(C),i))$.

By the definition above every nonempty subset C in a UPL concept language Lc has a least upper bound $lub(C)$ such that if an instance i is covered by at least one concept description $c \in C$, then it is covered by $lub(C)$ as well. Thus, the extensional representation of a concept given by $lub(C)$ is the union of the extensional representations of the concepts given by the descriptions in C. This explains the name of the UPL languages.

The IPL and UPL concept languages imply four groups of properties that explain the relations between the boundary sets of version spaces to be learned and version spaces with respect to positive and negative instances. These properties are used by the IBBS algorithms of the operations *Collapsed?, Classify, Subset?* and *Equal?*.

We start with the first group of properties. They are as follows:

(1.1) if a concept language Lc is IPL, then the minimal boundary set S of every nonempty version space VS, defined in Lc, has only one element;

(1.2) if a concept language Lc is UPL, then the maximal boundary set G of every nonempty version space VS, defined in Lc, has only one element.

Property (1.1) is proven in theorem 6.11; property (1.2) is proven in theorem 6.12.

Theorem 6.11. *If VS is the version space of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ such that $VS \neq \emptyset$ and the concept language Lc is IPL then:*

$$S = \{glb(VS)\}.$$

Proof. By definition of version spaces we have that $(\forall c \in VS) cons(c, \langle I^+, I^- \rangle)$. This implies according to definition 2.22:

$$(\forall p \in I^+)(\forall c \in VS) M(c, p) \tag{6.5}$$
$$(\forall n \in I^-)(\forall c \in VS) \neg M(c, n) \tag{6.6}$$

Since Lc is IPL using definition 6.9 $VS \subseteq Lc$ and $VS \neq \emptyset$ imply:

$$glb(VS) \in Lc. \tag{6.7}$$

Thus, according to definition 6.9 formulas (6.5) and (6.6) imply:

$$(\forall p \in I^+) M(glb(VS), p) \tag{6.8}$$
$$(\forall n \in I^-) \neg M(glb(VS), n) \tag{6.9}$$

But, according to definition 2.22 formulas (6.8) and (6.9) are equivalent to:

$$cons(glb(VS), \langle I^+, I^- \rangle). \tag{6.10}$$

Thus, from (6.7) and (6.10) using definition 3.1 it follows that $glb(VS) \in VS$. This means according to the definition of the greatest lower bound that $glb(VS)$ is the least element of the version space VS. Since $glb(VS)$ is the least element of the version space VS, $glb(VS)$ is the unique minimal element of VS (Birkoff, 1973). Therefore, $S = MIN(VS) = \{glb(VS)\}$. □

Theorem 6.12. *If VS is the version space of a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ such that $VS \neq \emptyset$ and the concept language Lc is UPL then:*

$$G = \{lub(VS)\}.$$

Proof. The proof is analagous to that of theorem 6.11. □

The second group of properties are consequences of theorems 6.11 and 6.12. They are as follows:

(2.1) if a concept language Lc is IPL, then the minimal boundary sets of two nonempty version spaces VS_1 and VS_2 in Lc are equal when they are defined for the same set of positive instances.

(2.2) if a concept language Lc is UPL, then the maximal boundary sets of two nonempty version spaces VS_1 and VS_2 in Lc are equal when they are defined for the same set of negative instances.

Property (2.1) is proven in corollary 6.13; property (2.2) is proven in corollary 6.14.

Corollary 6.13. *Consider the nonempty version space VS_1 of a task $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$, and the nonempty version space VS_2 of a task $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$. If $I_1^+ = I_2^+$ and the concept language Lc is IPL then:*

$$S_1 = S_2$$

where $S_1 = MIN(VS_1)$
$S_2 = MIN(VS_2)$.

Proof. The proof follows from theorem 6.11 and corollary A.4 in Appendix A. □

Corollary 6.14. *Consider the nonempty version space VS_1 of a task $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$, and the nonempty version space VS_2 of a task $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$. If $I_1^- = I_2^-$ and the concept language Lc is UPL then:*

$$G_1 = G_2$$

where $G_1 = MAX(VS_1)$
$G_2 = MAX(VS_2)$.

Proof. The proof follows from theorem 6.12 and corollary A.3 in Appendix A. □

The third group of properties follows from corollaries 6.13 and 6.14. They are as follows:

(3.1) if the concept language Lc is IPL, the version space VS to be learned and the set I^- of negative instances are not empty, then the minimal boundary sets S and $S(n)$ of the version spaces VS and $VS(n)$ are equal;

(3.2) if the concept language Lc is UPL, the version space VS to be learned and the set I^+ of positive instances are not empty, then the maximal boundary sets G and $G(p)$ of the version spaces VS and $VS(p)$ are equal.

Property (3.1) is proven in lemma 6.15; property (3.2) is proven in lemma 6.16.

Lemma 6.15. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language Lc is IPL,* $VS \neq \emptyset$, *and* $I^- \neq \emptyset$, *then:*

$$(\forall n \in I^-)(S(n) = S).$$

Proof. Consider arbitrary $n \in I^-$. By corollary 6.4 $VS \neq \emptyset$ implies that $VS(n) \neq \emptyset$. According to definitions 3.1 and 4.13 VS and $VS(n)$ are defined on the same set I^+ of positive instances. Thus, since Lc is IPL by corollary 6.13 it follows that $S(n) = S$. □

Lemma 6.16. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language Lc is UPL,* $VS \neq \emptyset$, *and* $I^+ \neq \emptyset$, *then:*

$$(\forall n \in I^+)(G(p) = G).$$

Proof. The proof is dual to that of lemma 6.15. □

The fourth group of properties is proven using theorems 6.11 and 6.12, corollaries 6.13 and 6.14, and lemmas 6.15 and 6.16. The properties are as follows:

(4.1) if the concept language Lc is IPL, and the set I^- of negative instances is not empty, then the minimal boundary set S of the version space VS to be learned is not empty if and only if the minimal boundary sets $S(n)$ of the version spaces $VS(n)$ with respect to negative instances are not empty;

(4.2) if the concept language Lc is UPL, and the set I^+ of positive instances is not empty, then the maximal boundary set G of the version space VS to be learned is not empty if and only if the maximal boundary sets $G(p)$ of the version spaces $VS(p)$ with respect to positive instances are not empty.

Property (4.1) is proven in theorem 6.17; property (4.2) is proven in theorem 6.18.

Theorem 6.17. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language Lc is IPL, and* $I^- \neq \emptyset$, *then:*

$$(S \neq \emptyset) \leftrightarrow (\forall n \in I^-)(S(n) \neq \emptyset).$$

Proof. ($\rightarrow$) Since Lc is IPL, Lc is admissible. Thus, by theorem A.7 $S \neq \emptyset$ implies that $VS \neq \emptyset$. By lemma 6.15 the fact that Lc is IPL, $VS \neq \emptyset$, and $I^- \neq \emptyset$ imply that $(\forall n \in I^-)(S(n) = S)$. But, $S \neq \emptyset$. Thus, $(\forall n \in I^-)(S(n) \neq \emptyset)$.

($\leftarrow$) Since Lc is IPL, Lc is admissible. Thus, by theorem A.7 $(\forall n \in I^-)(S(n) \neq \emptyset)$ and $I^- \neq \emptyset$ imply that $(\forall n \in I^-)(VS(n) \neq \emptyset)$. By theorem 6.11 each version space $VS(n)$ has exactly one minimal element $glb(VS(n))$. According to definition 4.13 each version space $VS(n)$ is defined on the same set I^+ of positive instances. Thus, by corollary 6.13 the only minimal elements of all the version spaces $VS(n)$ are equal; i.e., there exists $s \in Lc$ such that $(\forall n \in I^-)(S(n) = \{s\})$. This implies according to definition 4.13 that $(\forall n \in I^-)cons(s, \langle I^+, \{n\}\rangle)$. Using definition 2.22 we conclude that $(\forall n \in I^-)((\forall p \in I^+)M(c,p) \wedge \neg M(c,n))$. Thus, since $I^- \neq \emptyset$ we have that $((\forall p \in I^+)M(c,p) \wedge (\forall n \in I^-)\neg M(c,n))$. According to definition 2.22 this is equivalent to $cons(s, \langle I^+, I^-\rangle)$. By definition of version spaces $cons(s, \langle I^+, I^-\rangle)$ and $s \in Lc$ imply that $s \in VS$; i.e., $VS \neq \emptyset$. Thus, by theorem A.7 $S \neq \emptyset$. □

Theorem 6.18. *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$. If the concept language Lc is UPL, and $I^+ \neq \emptyset$, then:*

$$(G \neq \emptyset) \leftrightarrow (\forall p \in I^+)(G(p) \neq \emptyset).$$

Proof. The proof is dual to that of theorem 6.17. □

6.4 Algorithms of the Basic Version-Space Operations

This section considers algorithms of the basic version-space operations that are based on IBBS. The IBBS algorithms are given with their corresponding proofs and explanations.

6.4.1 Algorithms of the Search Operations

6.4.1.1 Functions for Computing Boundary Sets

The algorithms of the search operations *Initialise*, *Update*, and *Retraction* need efficient functions for computing the minimal and maximal boundary sets of version spaces $VS(p)$ and $VS(n)$. In this subsection we propose two such functions given in figures 6.2 and 6.3.

The first function *Generate-and-Prune-S* is used for generating the minimal boundary sets of version spaces with respect to positive training instances (see figure 6.2). It computes the set $S(p) = MIN(\{c \in Lc | cons(c, \langle\{p\}, I^-\rangle)\})$ given a positive training instance $p \in Li$ and a negative training set $I^- \subseteq Li$. The computation is realised in two steps and it is based on corollary A.4 (see Appendix A).

In the first step a minimal boundary set S_p of an auxiliary version space $VS_p = \{c \in Lc | M(c,p)\}$ is generated. The version space VS_p is a superset of the version space $VS(p) = \{c \in Lc | cons(c, \langle\{p\}, I^-\rangle)\}$ of which the minimal boundary set $S(p)$

Function *Generate-and-Prune-S*

Input: p: a positive training instance.
I^-: a set of negative training instances.
Lc: a concept language.

Output: $S(p) = MIN(\{c \in Lc | cons(c, \langle\{p\}, I^-\rangle)\})$.

$S_p = MIN(\{c \in Lc | M(c,p)\})$
$S(p) = \{s \in S_p | (\forall n \in I^-) \neg M(s,n)\}$
return $S(p)$.

Figure 6.2: The Function *Generate-and-Prune-S*.

Function *Generate-and-Prune-G*

Input: I^+: a set of positive training instances.
n: a negative training instance.
Lc: a concept language.

Output: $G(n) = MAX(\{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\})$.

$G_n = MAX(\{c \in Lc | \neg M(c,n)\})$
$G(n) = \{g \in G_n | (\forall p \in I^+) M(g,p)\}$
return $G(n)$.

Figure 6.3: The Function *Generate-and-Prune-G*.

has to be computed. Thus, by corollary A.4 applied to the version spaces VS_p and $VS(p)$ we have that $S(p) = \{s \in S_p | (\forall n \in I^-) \neg M(s,n)\}$. This means that in the second step the minimal boundary set $S(p)$ is formed from those elements of the set S_p that are consistent with the set I^-.

The second function *Generate-and-Prune-G* is used for generating the maximal boundary sets of version spaces with respect to negative training instances (see figure 6.3). It computes the set $G(n) = MAX(\{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\})$ given a positive training set $I^+ \subseteq Li$ and a negative training instance $n \in Li$. The function is based on corollary A.3 (see Appendix A) and thus it is dual to the function *Generate-and-Prune-S*.

6.4.1.2 Algorithm of the Operation *Initialise*

The operation *Initialise* sets the version space VS of a target concept when no training instances are available. The algorithm of the operation for the IBBS is proposed for the class of admissible concept languages since the IBBS are a correct representation for this class. It initialises the S-part and G-parts of the IBBS of the version space VS as empty families of sets.

6.4.1.3 Algorithm of the Operation *Update*

The operation *Update* revises the version space of a target concept when a new instance is added to the training sets. The algorithm of the operation for the IBBS has two procedures for handling positive and negative training instances, respectively. The procedures are based on theorems 6.19 and 6.20 below. The theorems are correct for the class of admissible concept languages. Thus, the algorithm is correct for this class as well.

Theorem 6.19. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space VS represented by IBBS:*$\langle \{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-} \rangle$, *and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with version space VS′ represented by IBBS:*$\langle \{S'(p)\}_{p\in I^{+\prime}}, \{G'(n)\}_{n\in I^-} \rangle$, *where* $I^{+\prime} = I^+ \cup \{\mathfrak{p}\}$. *If the concept language Lc is admissible then:*

$$
\begin{aligned}
&G'(n) = \{g \in G(n) \,|\, M(g, \mathfrak{p})\} \text{ for all } n \in I^- \\
&S'(p) = S(p) \text{ for all } p \in I^+ \\
&S'(\mathfrak{p}) = MIN(\{c \in Lc \,|\, cons(c, \langle \{\mathfrak{p}\}, I^- \rangle)\}).
\end{aligned}
$$

Proof. We prove the first part of the theorem. By lemma 6.2 for all $n \in I^-$:

$$
\begin{aligned}
VS(n) &= \{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\} \\
VS'(n) &= \{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\}.
\end{aligned}
$$

Thus, since $I^{+\prime} = I^+ \cup \{\mathfrak{p}\}$, by theorem A.1 (see Appendix A):

$$G'(n) = \{g \in G(n) \,|\, M(g, \mathfrak{p})\}.$$

We prove the second part of the theorem. By lemma 6.1 for all $p \in I^+$:

$$
\begin{aligned}
VS(p) &= \{c \in Lc | cons(c, \langle \{p\}, I^- \rangle)\} \\
VS'(p) &= \{c \in Lc | cons(c, \langle \{p\}, I^- \rangle)\}.
\end{aligned}
$$

Thus, $VS(p) = VS'(p)$, which implies according to definition 5.17 that:

$$S'(p) = S(p).$$

We prove the third part of the theorem. By lemma 6.1 the version space $VS'(\mathfrak{p})$ with respect to the instance $\mathfrak{p}$ is defined as follows:

$$VS'(\mathfrak{p}) = \{c \in Lc | cons(c, \langle\{\mathfrak{p}\}, I^-\rangle)\}.$$

Thus, according to definition 5.17:

$$\begin{aligned} S'(\mathfrak{p}) &= MIN(VS'(\mathfrak{p})) \\ &= MIN(\{c \in Lc | cons(c, \langle\{\mathfrak{p}\}, I^-\rangle)\}) \end{aligned}$$

The theorem is proven. □

Theorem 6.20. *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with version space VS represented by IBBS:$\langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle$, and a second task $\langle Li, Lc, M, \langle I^+, I^{-\prime}\rangle\rangle$ with version space VS' represented by IBBS:$\langle\{S'(p)\}_{p\in I^+}, \{G'(n)\}_{n\in I^{-\prime}}\rangle$, where $I^{-\prime} = I^- \cup \{\mathfrak{n}\}$. If the concept language Lc is admissible then:*

$$\begin{aligned} &S'(p) = \{s \in S(p) \mid \neg M(s, \mathfrak{n})\} \text{ for all } p \in I^+ \\ &G'(n) = G(n) \text{ for all } n \in I^- \\ &G'(\mathfrak{n}) = MAX(\{c \in Lc \mid cons(c, \langle I^+, \{\mathfrak{n}\}\rangle)\}). \end{aligned}$$

Proof. The proof is dual to that of theorem 6.19. □

The algorithm of the operation *Update* for the IBBS is given in figure 6.4. Applying the algorithm requires that the version space VS to be learned has been initialised.

When a positive instance $\mathfrak{p}$ is given the algorithm revises the G-part and the S-part of the IBBS. Revising the G-part the algorithm removes all elements of the maximal boundary sets $G(n)$ that do not cover the instance $\mathfrak{p}$. This guarantees that the new updated maximal boundary sets $G'(n)$ are the maximal boundary sets of the version spaces $VS'(n)$. Hence, the G-part of the updated version space VS' is correct according to theorem 6.19.

Revising the S-part the algorithm does not change the minimal boundary sets $S(p)$ of the version spaces $VS(p)$ for all $p \in I^+$. It generates the minimal boundary set $S'(\mathfrak{p})$ of the version space $VS'(\mathfrak{p})$ by the function *Generate-and-Prune-S*. The set $S'(\mathfrak{p})$ is set equal to all the minimal elements in the concept language Lc that are consistent with the positive instance $\mathfrak{p}$ and the set I^- of negative instances. The

Algorithm *Update*

Precondition: Lc: an admissible concept language.
Input: i: a new training instance.
IBBS:$\langle \{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-} \rangle$ of a version space VS.
Output: IBBS:$\langle \{S'(p)\}_{p\in I^+\cup\{\mathfrak{p}\}}, \{G'(n)\}_{n\in I^-} \rangle$ of a version space VS' s.t.
$VS' = \{c \in VS | M(c,\mathfrak{p})\}$ if i is a positive instance $\mathfrak{p}$.
IBBS:$\langle \{S'(p)\}_{p\in I^+}, \{G'(n)\}_{n\in I^-\cup\{\mathfrak{n}\}} \rangle$ of a version space VS' s.t.
$VS' = \{c \in VS | \neg M(c,\mathfrak{n})\}$ if i is a negative instance $\mathfrak{n}$.

if instance i is a positive instance $\mathfrak{p}$ **then**
  for $n \in I^-$ **do**
    $G'(n) = \{g \in G(n) \mid M(g,\mathfrak{p})\}$
  for $p \in I^+$ **do**
    $S'(p) = S(p)$
  $S'(\mathfrak{p}) =$ *Generate-and-Prune-S*$(\mathfrak{p}, I^-, Lc)$
  return $\langle \{S'(p)\}_{p\in I^+\cup\{\mathfrak{p}\}}, \{G'(n)\}_{n\in I^-} \rangle$
if instance i is a negative instance $\mathfrak{n}$ **then**
  for $p \in I^+$ **do**
    $S'(p) = \{s \in S(p) \mid \neg M(s,\mathfrak{n})\}$
  for $n \in I^-$ **do**
    $G'(n) = G(n)$
  $G'(\mathfrak{n}) =$ *Generate-and-Prune-G*$(I^+, \mathfrak{n}, Lc)$
  return $\langle \{S'(p)\}_{p\in I^+}, \{G'(n)\}_{n\in I^-\cup\{\mathfrak{n}\}} \rangle$.

Figure 6.4: The Algorithm of the Operation *Update*.

generated set $S'(\mathfrak{p})$ is added to the S-part of the IBBS; i.e., the algorithm implicitly intersects the version space VS with the version space $VS'(\mathfrak{p})$. Hence, the S-part of the updated version space VS' is correct according to theorem 6.19.

The behaviour of the algorithm for a negative training instance is dual to that for a positive training instance; hence it is analogous in form.

6.4.1.4 Algorithm of the Operation *Retraction*

The operation *Retraction* revises the version space of a target concept when an instance is removed from the training sets. The algorithm of the operation for IBBS has two procedures for handling positive and negative training instances, respectively. The procedures are based on theorems 6.21 and 6.22 given below (Smirnov and Braspenning, 1998b). The theorems are correct for the class of admissible concept

languages. Thus, the algorithm is correct for this class as well.

Theorem 6.21. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space VS represented by IBBS:*$\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$, *and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with version space VS′ represented by IBBS:*$\langle \{S'(p)\}_{p \in I^{+\prime}}, \{G'(n)\}_{n \in I^-} \rangle$, *where* $I^{+\prime} = I^+ - \{\mathfrak{p}\}$ *and* $\mathfrak{p} \in I^+$. *If the concept language Lc is admissible then:*

$$G'(n) = MAX(\{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\}) \text{ for } n \in I^-$$
$$S'(p) = S(p) \text{ for } p \in I^{+\prime}.$$

Proof. By lemma 6.2 for all $n \in I^-$:

$$VS'(n) = \{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\}.$$

Thus, according to definition 5.17:

$$G'(n) = MAX(\{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\}).$$

By lemma 6.1 for all $p \in I^{+\prime}$: $VS'(p) = VS(p)$ and thus according to definition 5.17 $S'(p) = S(p)$. □

Theorem 6.22. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space VS represented by IBBS:*$\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$, *and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle\rangle$ *with version space VS′ represented by IBBS:*$\langle \{S'(p)\}_{p \in I^+}, \{G'(n)\}_{n \in I^{-\prime}} \rangle$, *where* $I^{-\prime} = I^- - \{\mathfrak{n}\}$ *and* $\mathfrak{n} \in I^-$. *If the concept language Lc is admissible then:*

$$S'(p) = MIN(\{c \in Lc | cons(c, \langle \{p\}, I^{-\prime} \rangle)\}) \text{ for } p \in I^+$$
$$G'(n) = G(n) \text{ for } n \in I^{-\prime}.$$

Proof. The proof is dual to that of theorem 6.21. □

The algorithm of the operation *Retraction* is given in figure 6.5. When a positive instance $\mathfrak{p}$ has to be retracted it rebuilds the G-part and revises the S-part of the IBBS of the version space VS to be learned. Rebuilding the G-part the algorithm recomputes the maximal boundary sets $G'(n)$ of the version spaces $VS'(n)$ by the function *Generate-and-Prune-G*. Each boundary set $G'(n)$ is formed as a set of maximal descriptions in the concept language Lc used that are consistent with the revised set $I^+ - \{\mathfrak{p}\}$ and the corresponding negative instances n. Hence, according to theorem 6.21 the resulting sets $G'(n)$ form the G-part of the IBBS that is correct for the version space VS' to be learned.

Algorithm *Retraction*

Input: i: a training instance s.t. $i \in I^+ \cup I^-$.
IBBS:$\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$ of a version space VS.
Lc: an admissible concept language.
I^+: a set of positive instances.
I^-: a set of negative instances.

Output:
IBBS:$\langle \{S'(p)\}_{p \in I^+ - \{\mathfrak{p}\}}, \{G'(n)\}_{n \in I^-} \rangle$ of a version space VS' s.t.
$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\} \rangle)\}$
if i is a positive instance $\mathfrak{p}$.
IBBS:$\langle \{S'(p)\}_{p \in I^+}, \{G'(n)\}_{n \in I^- - \{\mathfrak{n}\}} \rangle$ of a version space VS' s.t.
$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\} \rangle)\}$
if i is a negative instance $\mathfrak{n}$.

if instance i is a positive instance $\mathfrak{p} \in I^+$ **then**
  for $n \in I^-$ **do**
    $G'(n) = Generate\text{-}and\text{-}Prune\text{-}G(I^+ - \{\mathfrak{p}\}, n, Lc)$
  for $p \in I^+ - \{\mathfrak{p}\}$ **do**
    $S'(p) = S(p)$
  return $\langle \{S'(p)\}_{p \in I^+ - \{\mathfrak{p}\}}, \{G'(n)\}_{n \in I^-} \rangle$
if instance i is a negative instance $\mathfrak{n} \in I^-$ **then**
  for $p \in I^+$ **do**
    $S'(p) = Generate\text{-}and\text{-}Prune\text{-}S(p, I^- - \{\mathfrak{n}\}, Lc)$
  for $n \in I^- - \{\mathfrak{n}\}$ **do**
    $G'(n) = G(n)$
  return $\langle \{S'(p)\}_{p \in I^+}, \{G'(n)\}_{n \in I^- - \{\mathfrak{n}\}} \rangle$.

Figure 6.5: The Algorithm of the Operation *Retraction*.

Revising the S-part the algorithm does not change the minimal boundary sets $S(p)$ of the version space $VS(p)$ for $p \in I^+ - \{\mathfrak{p}\}$. It removes from the IBBS only the minimal boundary set $S(\mathfrak{p})$ associated with the instance $\mathfrak{p}$. Hence, according to theorem 6.21 the S-part of the IBBS becomes correct for the version space VS' to be learned.

The behaviour of the algorithm when a negative instance is retracted is dual to that when a positive instance is retracted. Hence, it is analogous in form.

The algorithm of the operation *Retraction* in figure 6.5 has one serious problem. It recomputes the maximal boundary sets of version spaces $VS(n)$ whenever

a positive instance is retracted; it recomputes the minimal boundary sets of version spaces $VS(p)$ whenever a negative instance is retracted. This implies that the overall time complexity becomes high. Therefore, we give a more efficient algorithm of the operation *Retraction* in the next subsection.

6.4.1.5 Efficient Algorithm of the Operation *Retraction*

One way to overcome the computational problem of the algorithm of the operation *Retraction* is to restrict the concept languages used and to propose a new algorithm for them. In this subsection we propose such an algorithm of the operation *Retraction* that can be applied for admissible concept languages when two dual constraints 6.23 and 6.24 hold (see below).

The constraint 6.23 imposes a restriction on each pair of version spaces $VS(p_1)$ and $VS(p_2)$ of positive instances p_1 and p_2. More precisely, the subset of the elements of the minimal boundary set $S(p_1)$ covering the instance p_2 has to be equal to the subset of the elements of the minimal boundary set $S(p_2)$ covering the instance p_1. By duality, the constraint 6.24 imposes a restriction on each pair of version spaces $VS(n_1)$ and $VS(n_2)$ of negative instances n_1 and n_2. More precisely, the subset of the elements of the maximal boundary set $G(n_1)$ not covering the instance n_2 has to be equal to the subset of the elements of the maximal boundary set $G(n_2)$ not covering the instance n_1.

Constraint 6.23. *Consider an instance language Li and an admissible concept language Lc. Then for all* $p_1, p_2 \in Li$*, and all* $I \subseteq Li - \{p_1, p_2\}$*:*

$$\{s \in S(p_1) | M(s, p_2)\} = \{s \in S(p_2) | M(s, p_1)\}$$

where $S(p_1) = MIN(\{c \in Lc | cons(c, \langle \{p_1\}, I \rangle)\})$
$S(p_2) = MIN(\{c \in Lc | cons(c, \langle \{p_2\}, I \rangle)\})$.

Constraint 6.24. *Consider an instance language Li and an admissible concept language Lc. Then for all* $n_1, n_2 \in I^-$*, and all* $I \subseteq Li - \{n_1, n_2\}$*:*

$$\{g \in G(n_1) | \neg M(g, n_2)\} = \{g \in G(n_2) | \neg M(g, n_1)\}$$

where $G(n_1) = MAX(\{c \in Lc | cons(c, \langle I, \{n_1\} \rangle)\})$
$G(n_2) = MAX(\{c \in Lc | cons(c, \langle I, \{n_2\} \rangle)\})$.

The constraints certainly restrict the concept languages used, but still hold for broad sub-classes of admissible concept languages. For example, we have found that the constraints hold for conjunctive and disjunctive languages with tree-structured attributes.

The algorithm of the operation *Retraction* based on constraints 6.23 and 6.24 is called *Retraction*^{E}, where the superscript E stands for "efficient". It has two procedures for handling positive and negative training instances, respectively. The

correctness of the procedures is proven in theorems 6.25 and 6.26 below for the class of admissible concept languages when constraints 6.23 and 6.24 hold.

Theorem 6.25. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space VS represented by IBBS:*$\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$, *and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with version space* VS' *represented by IBBS:*$\langle \{S'(p)\}_{p \in I^{+\prime}}, \{G'(n)\}_{n \in I^-} \rangle$, *where* $I^{+\prime} = I^+ - \{\mathfrak{p}\}$ *and* $\mathfrak{p} \in I^+$. *If the concept language Lc is admissible then:*

$$G'(n) = G(n) \cup \{g \in G'(\mathfrak{n}) \mid \neg M(g, n)\} \text{ for } n \in I^-$$
$$S'(p) = S(p) \text{ for } p \in I^{+\prime}$$

where $\mathfrak{n} = \mathfrak{p}$
$G'(\mathfrak{n}) = MAX(\{c \in Lc | cons(c, \langle I^{+\prime}, \{\mathfrak{n}\}\rangle)\})$.

Proof. We prove the first part of the theorem. By lemma 6.2 the version space $VS'(n)$ for each $n \in I^-$ is defined as follows:

$$VS'(n) = \{c \in Lc | cons(c, \langle I^{+\prime}, \{n\}\rangle)\}.$$

To preserve notation 4.14 the instance $\mathfrak{p}$ is denoted by $\mathfrak{n}$. In this case the maximal boundary set $G'(n)$ of $VS'(n)$ obeys the following equality:

$$G'(n) = \{g \in G'(n) | M(g, \mathfrak{n})\} \cup \{g \in G'(n) | \neg M(g, \mathfrak{n})\} \tag{6.11}$$

Consider the first set $\{g \in G'(n) | M(g, \mathfrak{n})\}$ in (6.11). Since Lc is admissible by theorem A.1:

$$G(n) = \{g \in G'(n) | M(g, \mathfrak{n})\} \tag{6.12}$$

where $G(n)$ is the maximal boundary set of the version space $VS(n)$ defined as follows:

$$VS(n) = \{c \in Lc | cons(c, \langle I^{+\prime} \cup \{\mathfrak{n}\}, \{n\}\rangle)\}.$$

Consider the second set $\{g \in G'(n) | \neg M(g, \mathfrak{n})\}$ in (6.11). Since Lc is admissible by constraint 6.24:

$$\{g \in G'(n) | \neg M(g, \mathfrak{n})\} = \{g \in G'(\mathfrak{n}) | \neg M(g, n)\} \tag{6.13}$$

where $G'(\mathfrak{n})$ is the maximal boundary set of $VS'(\mathfrak{n})$ defined as follows:

$$VS'(\mathfrak{n}) = \{c \in Lc | cons(c, \langle I^{+\prime}, \{\mathfrak{n}\}\rangle)\}.$$

From (6.11), (6.12) and (6.13) we conclude that:

$$G'(n) = G(n) \cup \{g \in G'(\mathfrak{n}) \mid \neg M(g, n)\}.$$

The second part of the theorem has been already proven for theorem 6.21. Thus, the theorem is proven. □

Theorem 6.26. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with version space VS represented by IBBS:*$\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$, *and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle\rangle$ *with version space VS′ represented by IBBS:*$\langle \{S'(p)\}_{p \in I^+}, \{G'(n)\}_{n \in I^{-\prime}} \rangle$, *where* $I^{-\prime} = I^- - \{\mathfrak{n}\}$ *and* $\mathfrak{n} \in I^-$. *If the concept language Lc is admissible then:*

$$S'(p) = S(p) \cup \{s \in S'(\mathfrak{p}) \mid M(s,p)\} \text{ for } p \in I^+$$
$$G'(n) = G(n) \text{ for } n \in I^{-\prime}$$

where $\mathfrak{p} = \mathfrak{n}$
$S'(\mathfrak{p}) = MIN(\{c \in Lc | cons(c, \langle \{\mathfrak{p}\}, I^{-\prime} \rangle)\})$.

Proof. The proof is based on constraint 6.23 and it is dual to that of theorem 6.25. □

The algorithm *Retraction*^{E} is given in figure 6.6. When a positive instance $\mathfrak{p}$ has to be retracted it revises the G-part and the S-part of the IBBS of the version space VS to be learned. Revising the G-part the algorithm transforms the maximal boundary sets $G'(n)$ of the version spaces $VS'(n)$. Transforming is realised in two steps using an instance $\mathfrak{n}$ equal to the instance $\mathfrak{p}$ labelled as negative. In the first step the algorithm computes the set $G'(\mathfrak{n})$ by the function *Generate-and-Prune-G*. The set is generated as a set of maximal descriptions in the concept language Lc used that are consistent with the set $I^+ - \{\mathfrak{n}\}$ and the instance $\mathfrak{n}$. Note that the set $G'(\mathfrak{n})$ is computed only once. In the second step for each instance $n \in I^-$ the maximal boundary set $G'(n)$ is formed as a union of two sets. The first one is the maximal boundary set $G(n)$ of the version space $VS(n)$. The second set is a set of the elements of the set $G'(\mathfrak{n})$ that are consistent with the instance n. Hence, according to theorem 6.25 the resulting sets $G'(n)$ form the G-part of the IBBS that is correct with respect to the version space VS' to be learned.

The algorithm revises the S-part of IBBS of the version space VS analogously to the retraction algorithm given in figure 6.5.

The behaviour of the algorithm when a negative instance is retracted is dual to that of a positive instance being retracted. Hence, it is analogous in form.

The algorithm *Retraction*^{E} is efficient when it is compared with its predecessor from the previous subsection. This is due to the fact that the minimal and maximal boundary sets $S(p)$ and $G(n)$ do not have to be re-computed from scratch when training instances are retracted. More precisely:

- When a positive instance $\mathfrak{p}$ is retracted the set $G'(\mathfrak{n})$ is computed only once (where $\mathfrak{n} = \mathfrak{p}$). Each maximal boundary set $G'(n)$ is formed equal to its previous version $G(n)$ plus those elements of the set $G'(\mathfrak{n})$ that do not cover the instance n.

Algorithm *Retraction*^{E}

Input: i: a training instance s.t. $i \in I^+ \cup I^-$.
IBBS:$\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$ of a version space VS.
Lc: an admissible concept language so that constraints 6.23 and 6.24 hold.
I^+: a set of positive instances.
I^-: a set of negative instances.

Output:
IBBS:$\langle \{S'(p)\}_{p \in I^+ - \{\mathfrak{p}\}}, \{G'(n)\}_{n \in I^-} \rangle$ of a version space VS' s.t.
$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\} \rangle)\}$
if i is a positive instance $\mathfrak{p}$.
IBBS:$\langle \{S'(p)\}_{p \in I^+}, \{G'(n)\}_{n \in I^- - \{\mathfrak{n}\}} \rangle$ of a version space VS' s.t.
$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\} \rangle)\}$
if i is a negative instance $\mathfrak{n}$.

if instance i is a positive instance $\mathfrak{p} \in I^+$ **then**
  Let $\mathfrak{n}$ be the instance $\mathfrak{p}$ labelled as negative
  $G'(\mathfrak{n}) =$ *Generate-and-Prune-G*$(I^+ - \{\mathfrak{n}\}, \mathfrak{n}, Lc)$
  for $n \in I^-$ **do**
    $G'(n) = G(n) \cup \{g \in G'(\mathfrak{n}) \mid \neg M(g, n)\}$
  for $p \in I^+ - \{\mathfrak{p}\}$ **do**
    $S'(p) = S(p)$
  return $\langle \{S'(p)\}_{p \in I^+ - \{\mathfrak{p}\}}, \{G'(n)\}_{n \in I^-} \rangle$
if instance i is a negative instance $\mathfrak{n} \in I^-$ **then**
  Let $\mathfrak{p}$ be the instance $\mathfrak{n}$ labelled as positive
  $S'(\mathfrak{p}) =$ *Generate-and-Prune-S*$(\mathfrak{p}, I^- - \{\mathfrak{p}\}, Lc)$
  for $p \in I^+$ **do**
    $S'(p) = S(p) \cup \{s \in S'(\mathfrak{p}) \mid M(s, p)\}$
  for $n \in I^- - \{\mathfrak{n}\}$ **do**
    $G'(n) = G(n)$
  return $\langle \{S'(p)\}_{p \in I^+}, \{G'(n)\}_{n \in I^- - \{\mathfrak{n}\}} \rangle$.

Figure 6.6: The Efficient Algorithm of the Operation *Retraction*.

- When a negative instance $\mathfrak{n}$ is retracted the set $S'(\mathfrak{p})$ is computed only once (where $\mathfrak{p} = \mathfrak{n}$). Each minimal boundary set $S'(p)$ is formed equal to its previous version $S(p)$ plus those elements of the set $S'(\mathfrak{p})$ that do cover the instance p.

Hence, the overall time complexity of the algorithm is lower than that of its

predecessor. This is confirmed by the complexity analysis of the IBBS in section 6.5.

The algorithm $Retraction^E$ is more efficient in a comparison with the retraction algorithms of the boundary sets or the unilateral boundary sets. This is due to the fact that it does not require computing the minimal and maximal boundary sets of the auxiliary version spaces $\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}$ and $\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\}$ when the positive instances $\mathfrak{p}$ and the negative instances $\mathfrak{n}$ are retracted. Therefore, we conclude that the IBBS overcome the retraction problem of the boundary sets and the unilateral boundary sets.

6.4.2 Algorithms of the State-Test Operations

6.4.2.1 Algorithm of the Operation *Collapsed?*

The operation *Collapsed?* determines whether the version space VS of a target concept is empty. The algorithm of the operation for IBBS is designed for IPL and UPL concept languages only. This is due to two dual properties of these languages that we have not found in other sub-classes of admissible concept languages. The properties are as follows.

(P1) If the concept language is IPL and the negative training set I^- is not empty, then the version space VS to be learned is not empty if and only if all the version spaces $VS(n)$ with respect to negative instances are not empty.

(P2) If the concept language is UPL and the positive training set I^+ is not empty, then the version space VS to be learned is not empty if and only if all the version spaces $VS(p)$ with respect to positive instances are not empty.

The first property is proven in theorem 6.27 for IPL languages; the second property is proven in theorem 6.28 for UPL languages.

Theorem 6.27. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$. *If the concept language Lc is IPL and* $I^- \neq \emptyset$ *then:*

$$(VS \neq \emptyset) \leftrightarrow (\forall n \in I^-)(VS(n) \neq \emptyset).$$

Proof. The proof follows from theorems A.7 and 6.17. □

Theorem 6.28. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$. *If the concept language Lc is UPL and* $I^+ \neq \emptyset$ *then:*

$$(VS \neq \emptyset) \leftrightarrow (\forall p \in I^+)(VS(p) \neq \emptyset).$$

Proof. The proof follows from theorems A.8 and 6.18. □

Two simple consequences of theorems 6.27 and 6.28 are given below.

(C1) If the concept language is IPL and the negative training set I^- is not empty, then the version space VS to be learned is not empty if and only if all the maximal boundary sets $G(n)$ of the version spaces $VS(n)$ with respect to negative instances are not empty.

(C2) If the concept language is UPL and the positive training set I^+ is not empty, then the version space VS to be learned is not empty if and only if all the minimal boundary sets $S(p)$ of the version spaces $VS(p)$ with respect to positive instances are not empty.

The first consequence is reformulated in corollary 6.29; the second consequence in corollary 6.30.

Corollary 6.29. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language Lc is IPL and* $I^- \neq \emptyset$ *then:*

$$(VS \neq \emptyset) \leftrightarrow (\forall n \in I^-)(G(n) \neq \emptyset).$$

Proof. The proof follows from theorems A.8 and 6.27. □

Corollary 6.30. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language Lc is UPL and* $I^+ \neq \emptyset$ *then:*

$$(VS \neq \emptyset) \leftrightarrow (\forall p \in I^+)(S(p) \neq \emptyset).$$

Proof. The proof follows from theorems A.7 and 6.28. □

The algorithm of the operation *Collapsed?* for IPL languages is given in figure 6.7. It is based on the consequence C1. The input of the algorithm is a version space VS given by its IBBS representation. To test VS for a collapse the algorithm executes one of the steps given below.

- If the negative training set I^- is empty, then the algorithm forms the maximal boundary set $G = \{c \in MAX(Lc) | (\forall p \in I^+) M(c,p)\}$ of the version space VS[2]. If the set G is empty, then by theorem A.1 (see Appendix A) VS is empty (collapsed) and the algorithm returns true. Otherwise, VS is nonempty and the algorithm returns false.

[2] Note that the maximal boundary set G is initialised equal to $MAX(Lc)$. If the set I^- of negative instances is empty then by theorem A.1 $G = \{c \in MAX(Lc) | (\forall p \in I^+) M(c,p)\}$. See section 5.4 for more details.

Algorithm *IPL-Collapsed?*
Precondition: Lc: an IPL concept language.
Input: IBBS: $\langle \{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-} \rangle$ of a version space VS.
Output: true if $VS = \emptyset$.
false if $VS \neq \emptyset$.

if $I^- = \emptyset$ **then**
  $G = \{c \in MAX(Lc) | (\forall p \in I^+) M(c,p)\}$
  if $G = \emptyset$
    then return true
    else return false
if $(\exists n \in I^-)(G(n) = \emptyset)$ **then**
  return true
return false.

Figure 6.7: The Algorithm of the Operation *Collapsed?* for IPL languages.

- If the negative training set I^- is nonempty, then the algorithm uses the consequence C1. In this case the maximal boundary sets $G(n)$ from the IBBS representation of the version space VS are tested for a collapse. If at least one of them is empty, then by corollary 6.29 the version space VS is empty (collapsed) and the algorithm returns true. Otherwise, VS is not empty and the algorithm returns false.

The algorithm of the operation *Collapsed?* for UPL languages is given in figure 6.8. It is based on the consequence C2. The algorithm is dual to that for IPL languages. That is why its explanation is analogous.

The algorithm of the operation *Collapsed?*, that can be applied simultaneously for IPL and UPL languages, is given in figure 6.9. It is a union of the previous two algorithms and that is why its explanation coincides with their explanations.

6.4.2.2 No Algorithm of the Operation *Converged?*

The IBBS do not have an algorithm of the operation *Converged?*. The main reason for this is that the minimal and maximal boundary sets in the IBBS belong to the opposite levels of generality in the concept languages. Thus, the natural solution of the convergence problem, found for the boundary sets, does not exist for the IBBS.

There is one way to avoid this problem when computing the minimal boundary sets S of version spaces VS from IBBS is efficient for IPL languages. If so, then when the sets S have exactly one element s we can use a proposition given by Hirsh (1992b) (see section 5.5 and theorem B.6 in Appendix B)). The proposition states

Algorithm *UPL-Collapsed?*
Precondition: Lc: a UPL concept language.
Input: IBBS: $\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$ of a version space VS.
Output: true if $VS = \emptyset$.
false if $VS \neq \emptyset$.

if $I^+ = \emptyset$ **then**
$S = \{c \in MIN(Lc) | (\forall n \in I^-) \neg M(c, n)\}$
if $S = \emptyset$
then return true
else return false
if $(\exists p \in I^+)(S(p) = \emptyset)$ **then**
return true
return false.

Figure 6.8: The Algorithm of the Operation *Collapsed?* for UPL languages.

Algorithm *Collapsed?*
Precondition: Lc: IPL or UPL concept language.
Input: IBBS: $\langle \{S(p)\}_{p \in I^+}, \{G(n)\}_{n \in I^-} \rangle$ of a version space VS.
Output: true if $VS = \emptyset$.
false if $VS \neq \emptyset$.

if $I^- = \emptyset$ **then**
$G = \{c \in MAX(Lc) | (\forall p \in I^+) M(c, p)\}$
if $G = \emptyset$
then return true
else return false
if $I^+ = \emptyset$ **then**
$S = \{c \in MIN(Lc) | (\forall n \in I^-) \neg M(c, n)\}$
if $S = \emptyset$
then return true
else return false
if $(\exists n \in I^-)(G(n) = \emptyset) \vee (\exists p \in I^+)(S(p) = \emptyset)$ **then**
return true
return false.

Figure 6.9: The Algorithm of the Operation *Collapsed?* for IPL and UPL languages.

that if all minimal generalisations of the element s can be efficiently generated and each of them is not covered by any element of every maximal boundary set $G(n)$, then the element s is the only element of the version spaces VS.

By duality, a similar solution exists for UPL languages. We do not consider the proposed solutions as convergence algorithms for IBBS. Quite the contrary. Since the algorithms require explicit computation of the sets S or G, the IBBS actually become unilateral IBBS; i.e., the type of representation of version spaces is changed.

6.4.3 Algorithms of the Operation *Classify*

The operation *Classify* determines the classification of instances using the rule of the unanimous vote of descriptions in version spaces. We consider algorithms of this operation separately for IPL and UPL concept languages in subsections 6.4.3.1 and 6.4.3.2, respectively. The separation is due to the different role that the version spaces $VS(p)$ and $VS(n)$ play in determining the unanimous vote of descriptions in version spaces for IPL and UPL languages, respectively. In subsection 6.4.3.3 we overcome this problem by combining the classification algorithms for IPL and UPL languages into one that can be applied for both languages.

6.4.3.1 Algorithm for Intersection-Preserving Languages

If concept languages are IPL, we determine the unanimous vote of descriptions in the version space VS of a target concept using the version spaces $VS(n)$ with respect to negative instances. This is realised according to theorems 6.31 and 6.32 below. Theorem 6.31 states that an instance is covered by all the descriptions in the version space VS if and only if the descriptions of all the version space $VS(n)$ cover the instance. Analogously, theorem 6.32 states that an instance is not covered by all the descriptions in the version space VS if and only if there exists at least one version space $VS(n)$ of which all the descriptions do not cover the instance.

Theorem 6.31. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language* Lc *is IPL,* $VS \neq \emptyset$, *and* $I^- \neq \emptyset$, *then:*

$$(\forall i \in Li)((\forall c \in VS)M(c,i) \leftrightarrow (\forall n \in I^-)(\forall c \in VS(n))M(c,i)).$$

Proof. Consider arbitrary $i \in Li$. Then:

$(\forall c \in VS)M(c,i)$ iff (theorem A.10)
$(\forall s \in S)M(s,i)$ iff (lemma 6.15)
$(\forall n \in I^-)(\forall s \in S(n))M(s,i)$ iff (theorem A.10)
$(\forall n \in I^-)(\forall c \in VS(n))M(c,i)$

□

Theorem 6.32. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language* Lc *is IPL, and* $I^- \neq \emptyset$, *then:*

$$(\forall i \in Li)((\forall c \in VS)\neg M(c,i) \leftrightarrow (\exists n \in I^-)(\forall c \in VS(n))\neg M(c,i)).$$

Proof. Consider arbitrary $i \in Li$. Then[3]:

$(\forall c \in VS)\neg M(c,i)$ iff (corollary 3.16)
$\{c \in VS | M(c,i)\} = \emptyset$ iff (theorem 6.27)
$(\exists n \in I^-)\, VS(I^+ \cup \{i\}, n) = \emptyset$ iff (corollary 3.16)
$(\exists n \in I^-)(\forall c \in VS(I^+, n))\neg M(c,i)$

□

We recall that by theorem A.10 (theorem A.11) an instance is positive (negative) if and only if it is (not) covered by all the elements of the minimal (maximal) boundary set of a version space. Since the IBBS of version spaces contain the maximal boundary sets $G(n)$, but do not contain the minimal boundary sets $S(n)$ of version spaces $VS(n)$, the algorithm of the operation *Classify* for IPL concept languages is based on corollaries 3.15 and 3.16, and theorems 6.32 and A.11. Following definition 3.14 the algorithm has three procedures P0, P1 and P2.

(P0) The first procedure is the operation *Collapsed?*. It checks whether the version space VS to be learned is empty. If so, then according to definition 3.14 the algorithm returns "?". Otherwise, the procedure P1 is called.

(P1) The second procedure determines when the instance i to be classified is covered by all the descriptions in the version space VS. It is based on corollary 3.15 and it is indirectly implemented by the operations *Update* and *Collapsed?*. Hence, the procedure works as follows: it updates the version space VS with the instance i considered as negative. If VS collapses, then by corollary 3.15 all the descriptions in VS cover the instance and the procedure returns "+" according to definition 3.14. Otherwise, the procedure P2 is called.

(P2) The third procedure determines when the instance i to be classified is not covered by all the descriptions in the version space VS. The procedure has two sub-procedures P2.1 and P2.2. The sub-procedure P2.1 is called when the set I^- of the negative instances is empty; otherwise the sub-procedure P2.2 is called.

(P2.1) The first sub-procedure is based on corollary 3.16 and it is indirectly implemented by the operations *Update* and *Collapsed?*. Hence, the sub-procedure works as follows: it updates the version space VS with the

[3] Note that we use in the proof the complete notation 4.14.

instance i considered as positive. If VS collapses, then by corollary 3.16 all the descriptions in VS do not cover the instance and the procedure returns "$-$" according to definition 3.14.

(P2.2) The second sub-procedure is based on theorem 6.32. Since every version space $VS(n)$ is given with its maximal boundary set $G(n)$, the procedure works according to theorem A.11. If the instance i is not covered by all elements of at least one maximal boundary set $G(n)$, then all the descriptions in the version space VS do not cover the instance and the sub-procedure returns "$-$" according to definition 3.14.

If the negative classification of the instance i cannot be determined, the procedure P2 returns "?" according to definition 3.14.

6.4.3.2 Algorithm for Union-Preserving Languages

If concept languages are UPL, we determine the unanimous vote of descriptions in the version space VS of a target concept using the version spaces $VS(p)$ with respect to positive instances. This is realised according to theorems 6.33 and 6.34. Theorem 6.33 states that an instance is covered by all the descriptions in the version space VS if and only if there exists at least one version space $VS(p)$ of which all the descriptions do cover the instance. Analogously, theorem 6.34 states that an instance is not covered by all the descriptions in the version space VS if and only if the descriptions of all the version spaces $VS(p)$ do not cover the instance.

Theorem 6.33. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language* Lc *is UPL, and* $I^- \neq \emptyset$, *then:*

$$(\forall i \in Li)((\forall c \in VS)M(c,i) \leftrightarrow (\exists p \in I^+)(\forall c \in VS(p))M(c,i)).$$

Proof. The proof is dual to that of theorem 6.32. □

Theorem 6.34. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$. *If the concept language* Lc *is UPL,* $VS \neq \emptyset$, *and* $I^- \neq \emptyset$, *then:*

$$(\forall i \in Li)((\forall c \in VS)\neg M(c,i) \leftrightarrow (\forall p \in I^+)(\forall c \in VS(p))\neg M(c,i)).$$

Proof. The proof is dual to that of theorem 6.31. □

Since the IBBS of version spaces contain the minimal boundary sets $S(p)$, but do not contain the maximal boundary sets $G(p)$ of version spaces $VS(p)$, the algorithm of the operation *Classify* for UPL concept languages is based on theorems 6.33 and A.10, and corollaries 3.15 and 3.16. Following definition 3.14 the algorithm has three procedures, again called P0, P1 and P2.

(P0) The first procedure is the operation *Collapsed?*. It checks whether the version space VS to be learned is empty. If so, then according to definition 3.14 the algorithm returns "?". Otherwise, the procedure P1 is called.

(P1) The second procedure determines when the instance i to be classified is covered by all the descriptions in the version space VS. The procedure has two sub-procedures P1.1 and P1.2. The sub-procedure P1.1 is called when the set I^+ of the positive instances is empty; otherwise the sub-procedure P1.2 is called.

(P1.1) The first sub-procedure is based on corollary 3.15 and it is indirectly implemented by the operations *Update* and *Collapsed?*. Hence, the sub-procedure works as follows: it updates the version space VS with the instance i considered as negative. If VS collapses, then by corollary 3.15 all the descriptions in VS do cover the instance and the procedure returns "+" according to definition 3.14.

(P1.2) The second sub-procedure is based on theorem 6.33. Since every version space $VS(p)$ is given with its minimal boundary set $S(p)$ the procedure works according to theorem A.10 as follows: if the instance i is covered by all elements of at least one minimal boundary set $S(p)$, then all the descriptions in the version space VS do cover the instance and the procedure returns "+" according to definition 3.14.

If the procedure P1 cannot determine the positive classification of the instance i, the procedure P2 is called.

(P2) The third procedure determines when the instance i to be classified is not covered by all the descriptions in the version space VS. It is based on corollary 3.16 and is indirectly implemented by the operations *Update* and *Collapsed?*. Hence, the procedure works as follows: it updates the version space VS with the instance i considered as positive. If VS collapses, then by corollary 3.16 all the descriptions in VS do not cover the instance and the procedure returns "−" according to definition 3.14. Otherwise, "?" is returned; i.e., the classification of the instance cannot be determined.

6.4.3.3 Algorithm for Intersection-Preserving and Union-Preserving Languages

The algorithms of the operation *Classify* given in the previous two subsections are not symmetric. They contain one procedure, that is specific for the class of the concept languages used, and two procedures, that can be used for IPL and UPL languages. We use this observation to propose an algorithm of the operation *Classify* that can be used simultaneously for IPL and UPL concept languages. The algorithm contains the operation *Collapsed?*, the procedure P1 of the algorithm for IPL languages, and

Algorithm *Classify*

Precondition: Lc: IPL or UPL concept language.
Input: i: a instance to be classified.
IBBS:$\langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle$ of a version space VS.
Output: "+" if $(VS \neq \emptyset) \wedge (\forall c \in VS) M(c,i)$.
"−" if $(VS \neq \emptyset) \wedge (\forall c \in VS) \neg M(c,i)$.
"?" otherwise.

if $Collapsed?(\langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^- \cup \{i\}}\rangle)$ **then**
  return "?"
let $\mathfrak{n}$ be i labelled as negative
$\langle\{S'(p)\}_{p\in I^+}, \{G'(n)\}_{n\in I^- \cup \{\mathfrak{n}\}}\rangle = Update(\mathfrak{n}, \langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle)$
if $Collapsed?(\langle\{S'(p)\}_{p\in I^+}, \{G'(n)\}_{n\in I^- \cup \{\mathfrak{n}\}}\rangle)$ **then**
  return "+"
let $\mathfrak{p}$ be i labelled as positive
$\langle\{S'(p)\}_{p\in I^+ \cup \{\mathfrak{p}\}}, \{G'(n)\}_{n\in I^-}\rangle = Update(\mathfrak{p}, \langle\{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-}\rangle)$
if $Collapsed?(\langle\{S'(p)\}_{p\in I^+ \cup \{\mathfrak{p}\}}, \{G'(n)\}_{n\in I^-}\rangle)$ **then**
  return "−"
return "?".

Figure 6.10: The Algorithm of the Operation *Classify*.

the procedure P2 of the algorithm for UPL languages. Hence, the algorithm is based on corollaries 3.15 and 3.16; i.e., on the operations *Update* and *Collapsed?*.

The algorithm is given in figure 6.10. Its explanation coincides with the explanation of the procedure P0, the explanation of the procedure P1 of the classification algorithm for IPL languages, and the explanation of the procedure P2 of the classification algorithm for UPL languages.

6.4.4 Additional Set Operations

6.4.4.1 Algorithm of the Operation *Member?*

The operation *Member?* determines whether a description in a concept language Lc belongs to the version space VS of a target concept. The algorithm of the operation for the IBBS is based on theorem 6.8 and thus it is correct for the class of admissible concept languages. The algorithm is given in figure 6.11. Its input is a concept description $c \in Lc$ and the IBBS of a version space VS. To test membership of c in VS the algorithm checks whether there exists a minimal boundary set $S(p)$ in the IBBS which elements are not more specific than c. If so, by theorem 6.8 the

Algorithm *Member?*

Precondition: Lc: an admissible concept language.
Input: c: a concept description.
IBBS:$\langle \{S(p)\}_{p\in I^+}, \{G(n)\}_{n\in I^-} \rangle$ of a version space VS.
Output: true if $(c \in VS)$.
false if $\neg(c \in VS)$.

if $(\exists p \in I^+)(\forall s \in S(p))\neg(s \leq c)$ **then**
return false
if $(\exists n \in I^-)(\forall g \in G(n))\neg(c \leq g)$ **then**
return false
return true.

Figure 6.11: The Algorithm of the Operation *Member?*.

description c does not belong to the version space VS and the algorithm returns false. Otherwise, the algorithm checks whether there exists a maximal boundary set $G(n)$ in the IBBS which elements are not more general than the description c. If so, by theorem 6.8 the description c does not belong to the version space VS and the algorithm returns false. Otherwise, the description c does belong to the version space VS and the algorithm returns true.

6.4.4.2 Algorithm of the Operation *Intersection*

The operation *Intersection* intersects two version spaces and returns the resulting version space. The algorithm of the operation for the IBBS is based on theorem 6.35 and corollary 6.36.

Theorem 6.35 considers the case when we have three version spaces VS_1, VS_2 and VS_{12} such that $VS_{12} = VS_1 \cap VS_2$. It proposes a way for computing the minimal and maximal boundary sets $S_{12}(p)$ and $G_{12}(n)$ of version spaces $VS_{12}(p)$ and $VS_{12}(n)$ in terms of the boundary sets $S_1(p)$, $S_2(p)$, $G_1(n)$ and $G_2(n)$ of version spaces $VS_1(p)$, $VS_1(n)$, $VS_2(p)$ and $VS_2(n)$.

Theorem 6.35. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with version space* VS_1 *represented by IBBS:*$\langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-} \rangle$*; a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2 *represented by IBBS:*$\langle \{S_2(p)\}_{p\in I_2^+}, \{G_2(n)\}_{n\in I_2^-} \rangle$*; and a third task* $\langle Li, Lc, M, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^- \rangle\rangle$ *with version space* VS_{12} *represented by IBBS:* $\langle \{S_{12}(p)\}_{p\in I_1^+ \cup I_2^+}, \{G_{12}(n)\}_{n\in I_1^- \cup I_1^-} \rangle$*. If Lc is admissible then:*

$$S_{12}(p) = \begin{cases} \{s \in S_1(p) \,|\, (\forall n \in I_2^- - I_1^-)\neg M(s,n)\} & \text{if } p \in I_1^+ \\ \{s \in S_2(p) \,|\, (\forall n \in I_1^- - I_2^-)\neg M(s,n)\} & \text{if } p \in I_2^+. \end{cases}$$

$$G_{12}(n) = \begin{cases} \{g \in G_1(n) \,|\, (\forall p \in I_2^+ - I_1^+) M(g,p)\} & \text{if } n \in I_1^- \\ \{g \in G_2(n) \,|\, (\forall p \in I_1^+ - I_2^+) M(g,p)\} & \text{if } n \in I_2^-. \end{cases}$$

Proof. We prove the first part of the theorem.

1). Consider an arbitrarily chosen instance $p \in I_1^+$ and its corresponding version spaces $VS_1(p)$ and $VS_{12}(p)$. By lemma 6.1:

$$VS_1(p) = \{c \in Lc \,|\, cons(c, \langle\{p\}, I_1^-\rangle)\}$$
$$VS_{12}(p) = \{c \in Lc \,|\, cons(c, \langle\{p\}, I_1^- \cup I_2^-\rangle)\}$$

Thus, since $I_1^- \subseteq I_1^- \cup I_2^-$ and Lc is admissible by corollary A.4 (from Appendix A) we have that:

$$S_{12}(p) = \{s \in S_1(p) \,|\, (\forall n \in I_2^- - I_1^-)\neg M(s,n)\}.$$

2). Consider an arbitrarily chosen instance $p \in I_2^+$ and its corresponding version spaces $VS_2(p)$ and $VS_{12}(p)$. By lemma 6.1:

$$VS_2(p) = \{c \in Lc \,|\, cons(c, \langle\{p\}, I_2^-\rangle)\}$$
$$VS_{12}(p) = \{c \in Lc \,|\, cons(c, \langle\{p\}, I_1^- \cup I_2^-\rangle)\}$$

Thus, since $I_2^- \subseteq I_1^- \cup I_2^-$ and Lc is admissible by corollary A.4 (from Appendix A) we have that:

$$S_{12}(p) = \{s \in S_2(p) \,|\, (\forall n \in I_1^- - I_2^-)\neg M(s,n)\}.$$

The first part of the theorem is proven.

The proof of the second part of the theorem is dual to that of the first part.

□

In principle, theorem 6.35 can be used as a basis for the algorithm of the operation *Intersection* for the IBBS. However, the main problem of such an algorithm is computing twice the minimal and maximal boundary sets $S_{12}(p)$ and $G_{12}(n)$ for instances $p \in I_1^+ \cap I_2^+$ and $n \in I_1^- \cap I_2^-$. To overcome this problem we propose a simple corollary of the theorem.

Algorithm *Intersection*

Precondition: Lc: an admissible concept language.
Input: IBBS:$\langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-}\rangle$ of a version space VS_1.
IBBS:$\langle \{S_2(p)\}_{p\in I_2^+}, \{G_2(n)\}_{n\in I_2^-}\rangle$ of a version space VS_2.
Output: IBBS:$\langle \{S_{12}(p)\}_{p\in I_1^+\cup I_2^+}, \{G_{12}(n)\}_{n\in I_1^-\cup I_2^-}\rangle$ of a version space $VS_{12} = VS_1 \cap VS_2$.

for $p \in I_1^+$ **do**
  $S_{12}(p) = \{s \in S_1(p) \mid (\forall n \in I_2^- - I_1^-)\neg M(s,n)\}$
for $p \in I_2^+ - I_1^+$ **do**
  $S_{12}(p) = \{s \in S_2(p) \mid (\forall n \in I_1^- - I_2^-)\neg M(s,n)\}$
for $n \in I_1^-$ **do**
  $G_{12}(n) = \{g \in G_1(n) \mid (\forall p \in I_2^+ - I_1^+) M(g,p)\}$
for $n \in I_2^- - I_1^-$ **do**
  $G_{12}(n) = \{g \in G_2(n) \mid (\forall p \in I_1^+ - I_2^+) M(g,p)\}$
return $\langle \{S_{12}(p)\}_{p\in I_1^+\cup I_2^+}, \{G_1(n)\}_{n\in I_1^-\cup I_2^-}\rangle$.

Figure 6.12: The Algorithm of the Operation *Intersection*.

Corollary 6.36. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^-\rangle\rangle$ *with version space* VS_1 *represented by IBBS:*$\langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-}\rangle$*; a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^-\rangle\rangle$ *with version space* VS_2 *represented by IBBS:*$\langle \{S_2(p)\}_{p\in I_2^+}, \{G_2(n)\}_{n\in I_2^-}\rangle$*; and a third task* $\langle Li, Lc, M, \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^-\rangle\rangle$ *with version space* VS_{12} *represented by IBBS:* $\langle \{S_{12}(p)\}_{p\in I_1^+\cup I_2^+}, \{G_{12}(n)\}_{n\in I_1^-\cup I_1^-}\rangle$. *If Lc is admissible then:*

$$S_{12}(p) = \begin{cases} \{s \in S_1(p) \mid (\forall n \in I_2^- - I_1^-)\neg M(s,n)\} & \text{if } p \in I_1^+ \\ \{s \in S_2(p) \mid (\forall n \in I_1^- - I_2^-)\neg M(s,n)\} & \text{otherwise.} \end{cases}$$

$$G_{12}(n) = \begin{cases} \{g \in G_1(n) \mid (\forall p \in I_2^+ - I_1^+) M(g,p)\} & \text{if } n \in I_1^- \\ \{g \in G_2(n) \mid (\forall p \in I_1^+ - I_2^+) M(g,p)\} & \text{otherwise.} \end{cases}$$

Proof. The proof follows from theorem 6.35. □

The corollary determines completely the IBBS algorithm of the operation *Intersection* presented in figure 6.12. The algorithm is applicable for the whole class of admissible concept languages. Its input is the IBBS of version spaces VS_1 and VS_2 that are to be intersected; its output is the IBBS of the resulting version space VS_{12}.

The algorithm creates the minimal boundary sets $S_{12}(p)$ of version spaces $VS_{12}(p)$ in two steps. In the first step it forms the minimal boundary sets $S_{12}(p)$ for

positive instances $p \in I_1^+$. Each set $S_{12}(p)$ is generated equal to a subset of the corresponding minimal boundary set $S_1(p)$ which elements do not cover the instances of the set $I_2^- - I_1^-$. In the second step the algorithm forms the minimal boundary sets $S_{12}(p)$ for positive instances $p \in I_2^+ - I_1^+$. Each set $S_{12}(p)$ is generated equal to a subset of the corresponding minimal boundary set $S_1(p)$ which elements do not cover the instances of the set $I_1^- - I_2^-$. Thus, the algorithm avoids computing twice the sets $S_{12}(p)$ for positive instances $p \in I_1^+ \cap I_2^+$.

The algorithm creates the maximal boundary sets $G_{12}(p)$ of version spaces $VS_{12}(p)$ in two steps as well. In the first step it forms the maximal boundary sets $G_{12}(n)$ for negative instances $n \in I_1^-$. Each set $G_{12}(n)$ is generated equal to a subset of the corresponding maximal boundary set $G_1(n)$ which elements cover the instances of the set $I_2^+ - I_1^+$. In the second step the algorithm forms the maximal boundary sets $G_{12}(n)$ for negative instances $n \in I_2^- - I_1^-$. Each set $G_{12}(n)$ is generated equal to a subset of the corresponding maximal boundary set $G_1(n)$ which elements cover the instances of the set $I_1^+ - I_2^+$. Thus, the algorithm avoids computing twice the sets $G_{12}(n)$ for negative instances $n \in I_1^- \cap I_2^-$.

By corollary 6.36 the resulting minimal and maximal boundary sets $S_{12}(p)$ and $G_{12}(p)$ form the IBBS of the version space $VS_{12} = VS_1 \cap VS_2$. Hence, these IBBS are returned as a final result of the algorithm execution.

6.4.4.3 Algorithm of the Operation *Subset?*

The operation *Subset?* determines whether a version space is a subset of another version space. The algorithm of the operation for the IBBS is based on theorem 3.22 and thus it uses the operation *Classify*. The operation *Classify* has an algorithm for IPL and UPL concept languages only. Hence, the algorithm of the operation *Subset?* can be applied only for these languages.

The algorithm is given in figure 6.13. To test whether a version space VS_1 is a subset of another version space VS_2 the algorithm classifies each training instance of the sets I_2^+ and I_2^- using the version space VS_1. This is realised with the algorithm of the operation *Classify* applied on the IBBS of the version space VS_1. Thus, if the label of each instance in the sets I_2^+ and I_2^- is identical to the instance's classification with VS_1, then by theorem 3.22 the algorithm returns true; i.e., VS_1 is a subset of VS_2. Otherwise it returns false; i.e., VS_1 is not a subset of VS_2.

6.4.4.4 Algorithm of the Operation *Equal?*

The operation *Equal?* determines whether a version space VS_1 is equal to another version space VS_2. The algorithm of the operations for the IBBS is based on the set-equality definition from section 3.2.4.4. Hence, the algorithm is a sequence of two calls of the operation *Subset?* (see figure 6.14). The first call checks whether $VS_1 \subseteq VS_2$; the second call checks whether $VS_2 \subseteq VS_1$. If both calls return true,

Algorithm *Subset?*

Precondition: *Lc*: IPL or UPL concept language.
Input: IBBS:$\langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-} \rangle$ of a version space VS_1.
A pair of training sets $\langle I_2^+, I_2^- \rangle$ of a version space VS_2.
Output: true if $(VS_1 \subseteq VS_2)$.
false if $\neg(VS_1 \subseteq VS_2)$.

for $i \in I_2^+ \cup I_2^-$ **do**
if *Classify*$(i, \langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-} \rangle) \neq label(i)$
then return false
return true.

Figure 6.13: The Algorithm of the Operation *Subset?*.

Algorithm *Equal?*

Precondition: *Lc*: IPL or UPL concept language.
Input: A pair of training sets $\langle I_1^+, I_1^- \rangle$ defining a version space VS_1.
A pair of training sets $\langle I_2^+, I_2^- \rangle$ defining a version space VS_2.
IBBS:$\langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-} \rangle$ of a version space VS_1.
IBBS:$\langle \{S_2(p)\}_{p\in I_2^+}, \{G_2(n)\}_{n\in I_2^-} \rangle$ of a version space VS_2.
Output: true if $(VS_1 = VS_2)$.
false if $\neg(VS_1 = VS_2)$.

if $\neg$*Subset?*$(\langle \{S_1(p)\}_{p\in I_1^+}, \{G_1(n)\}_{n\in I_1^-} \rangle, \langle I_2^+, I_2^- \rangle)$ **then**
return false
if $\neg$*Subset?*$(\langle \{S_2(p)\}_{p\in I_2^+}, \{G_2(n)\}_{n\in I_2^-} \rangle, \langle I_1^+, I_1^- \rangle)$ **then**
return false
return true.

Figure 6.14: The Algorithm of the Operation *Equal?*.

then by the set-equality definition VS_1 is equal to VS_2 and the algorithm returns true. Otherwise, VS_1 is not equal to VS_2 and the algorithm returns false.

Since the operation *Subset?* is implemented for IPL and UPL languages only, the algorithm of the operation *Equal?* can be applied only for these languages.

6.5 Worst-Case Complexity Analysis

This section presents a worst-case complexity analysis of the IBBS representation and the IBBS algorithms of the basic version-space operations. The results of the analysis are given in terms of:

- the number P of positive instances;
- the number N of negative instances;
- the largest size Σ of the set $MIN(Lc)$;
- the largest size Γ of the set $MAX(Lc)$;
- the largest size Σ_p of the set $MIN(\{c \in Lc | M(c,p)\})$ for all $p \in Li$;
- the largest size Γ_n of the set $MAX(\{c \in Lc | \neg M(c,n)\})$ for all $n \in Li$;
- the worst-case time complexity $t^{\downarrow}$ for generating the set $MIN(Lc)$;
- the worst-case time complexity $t^{\uparrow}$ for generating the set $MAX(Lc)$;
- the worst-case time complexity $t_p^{\downarrow}$ for generating the set $MIN(\{c \in Lc | M(c,p)\})$ for all $p \in Li$;
- the worst-case time complexity $t_n^{\uparrow}$ for generating the set $MAX(\{c \in Lc | \neg M(c,n)\})$ for all $n \in Li$;
- the worst-case time complexity t_c to test whether one concept description covers an instance.

In the definitions of the largest sizes Σ, Γ Σ_p and Γ_n we assume that the concept language Lc belongs to either the class of admissible languages or the class of IPL and UPL languages. The concrete belonging depends on the algorithm under study. More precisely, it depends on the type of concept languages for which the correctness of the algorithm has been proven.

Since the complexity analysis is worst-case we determine the conditions of the worst case. They are as follows:

(1) the elements of the minimal boundary sets $S(p)$ do not cover any negative training instance; and

(2) the elements of the maximal boundary sets $G(n)$ do cover every positive training instance.

Condition (1) implies that the largest size of the minimal boundary sets $S(p)$ of version spaces $VS(p)$ is equal to the size Σ_p. By duality, condition (2) implies that the largest size of the maximal boundary sets $G(n)$ of version spaces $VS(n)$ is equal to the size Γ_n. Thus, in the analysis we substitute the sizes of the minimal and maximal boundary sets $S(p)$ and $G(n)$ with Σ_p and Γ_n, respectively.

6.5.1 Worst-Case Space-Complexity Analysis of the Representation

The worst-case space complexity of the IBBS is equal to the sum of the space complexities of P minimal boundary sets $S(p)$ with size Σ_p and N maximal boundary sets $G(n)$ with size Γ_n. Thus, it is equal to $O(P\Sigma_p + N\Gamma_n)$.

6.5.2 Worst-Case Time-Complexity Analysis of the Algorithms

This subsection presents a worst-case time-complexity analysis of the algorithms of the basic version-space operations based on the IBBS. The results are summarised in table 6.1.

Algorithms	Time Complexity
Initialise	$O(1)$
Update ($\oplus$ instance)	$O(t_p^{\downarrow}+N(\Sigma_p+\Gamma_n)t_c)$
Update ($\ominus$ instance)	$O(t_n^{\uparrow}+P(\Sigma_p+\Gamma_n)t_c)$
Retraction ($\oplus$ instance)	$O(N(t_n^{\uparrow}+P\Gamma_n t_c))$
Retraction ($\ominus$ instance)	$O(P(t_p^{\downarrow}+N\Sigma_p t_c))$
Retraction^E ($\oplus$ instance)	$O(t_n^{\uparrow}+(P+N)\Gamma_n t_c)$
Retraction^E ($\ominus$ instance)	$O(t_p^{\downarrow}+(P+N)\Sigma_p t_c)$
Collapsed?	$O(t^{\downarrow}+t^{\uparrow}+(P\Gamma+N\Sigma)t_c)$
Classify ($\oplus$ instance)	$O(t^{\downarrow}+t^{\uparrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+P(\Sigma_p+\Gamma_n)t_c)$
Classify ($\ominus$ instance)	$O(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+(P\Gamma+N\Sigma)t_c+N(\Sigma_p+\Gamma_n)t_c)$
Classify	$O(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+(P+N)(\Sigma_p+\Gamma_n)t_c)$
Member?	$O(P\Sigma_p t_c+N\Gamma_n t_c)$
Intersection	$O(PN(\Sigma_p+\Gamma_n)t_c)$
Subset?	$O(P(t^{\downarrow}+t^{\uparrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+P(\Sigma_p+\Gamma_n)t_c)+$ $N(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+(P\Gamma+N\Sigma)t_c+N(\Sigma_p+\Gamma_n)t_c))$
Equal?	$O(P(t^{\downarrow}+t^{\uparrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+P(\Sigma_p+\Gamma_n)t_c)+$ $N(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+(P\Gamma+N\Sigma)t_c+N(\Sigma_p+\Gamma_n)t_c))$

Table 6.1: Worst-Case Time Complexities of the IBBS Algorithms of the Basic Version-Space Operations.

6.5.2.1 The Functions *Generate-and-Prune-S* and *Generate-and-Prune-G*

The worst-case time complexity of the function *Generate-and-Prune-S* is equal to $O(t_p^{\downarrow} + N\Sigma_p t_c)$ (see figure 6.2). The term $O(t_p^{\downarrow})$ is the worst-case complexity for

generating the set S_p. The term $O(N\Sigma_p t_c)$ arises because Σ_p elements of the set S_p are checked whether they cover N negative instances.

The worst-case time complexity of the function *Generate-and-Prune-G* can be derived by duality. It is equal to $O(t_n^{\uparrow} + P\Gamma_n t_c)$ (see figure 6.3).

We use the derived time complexities of the functions in a time complexity analysis of the algorithms of the search operations (see below).

6.5.2.2 The Algorithm of the Operation *Initialise*

The worst-case time complexity of the IBBS algorithm of the operation *Initialise* is equal to $O(1)$. This is due to the fact that the algorithm initialises the S-part and G-part of the IBBS equal to empty families of sets.

6.5.2.3 The Algorithm of the Operation *Update*

The IBBS algorithm of the operation *Update* consists of two procedures for handling positive and negative training instances, respectively. Hence, the time complexity analysis of the algorithm is realised for each of the procedures separately.

The worst-case time complexity of the procedure for processing one positive instance $\mathfrak{p}$ is equal to the time complexity $O(N\Gamma_n t_c)$ for updating the sets $G(n)$ plus the time complexity for generating the set $S'(\mathfrak{p})$ (see figure 6.4). Since generating the set $S'(\mathfrak{p})$ is realised by the function *Generate-and-Prune-S*, its worst-case time complexity is $O(t_p^{\downarrow} + N\Sigma_p t_c)$. Thus, the worst case time complexity of the procedure is $O(N\Gamma_n t_c + t_p^{\downarrow} + N\Sigma_p t_c) = O(t_p^{\downarrow} + N(\Sigma_p + \Gamma_n)t_c)$.

The worst-case time complexity of the procedure for processing one negative instance $\mathfrak{n}$ is derived by duality. It is $O(t_n^{\uparrow} + P(\Sigma_p + \Gamma_n)t_c)$.

6.5.2.4 The Algorithm of the Operation *Retraction*

The IBBS algorithm of the operation *Retraction* consists of two procedures for handling positive and negative training instances, respectively. Thus, the time complexity analysis of the algorithm is realised for each of the procedures separately.

The worst-case time complexity of the procedure for processing one positive instance $\mathfrak{p}$ is equal to the time complexity of generating the sets $G'(n)$ (see figure 6.5). Since generating each set $G'(n)$ is realised by the function *Generate-and-Prune-G*, it takes $O(t_n^{\uparrow} + P\Gamma_n t_c)$ in the worst case. Thus, the worst case time complexity of the procedure is $O(N(t_n^{\uparrow} + P\Gamma_n t_c))$.

The worst-case time complexity of the procedure for processing one negative instance can be derived by duality. It is $O(P(t_p^{\downarrow} + N\Sigma_p t_c))$.

6.5.2.5 The Efficient Algorithm of the Operation *Retraction*

The efficient IBBS algorithm of the operation *Retraction* consists of two procedures for handling positive and negative training instances, respectively. Thus, the time

complexity analysis of the algorithm is realised for each of the procedures separately.

The worst-case time complexity of the procedure for processing one positive instance $\mathfrak{p}$ is equal to the time complexity for generating the set $G'(\mathfrak{n})$ plus the time complexity for updating the sets $G'(n)$, where $\mathfrak{n} = \mathfrak{p}$ (see figure 6.6). The set $G'(\mathfrak{n})$ is generated by the function *Generate-and-Prune-G* that takes $O(t_n^{\uparrow} + P\Gamma_n t_c)$. Updating the sets $G'(n)$ takes $O(N\Gamma_n t_c)$. Thus, the worst-case time complexity of the procedure is $O(t_n^{\uparrow} + P\Gamma_n t_c + N\Gamma_n t_c) = O(t_n^{\uparrow} + (P+N)\Gamma_n t_c)$.

The worst-case time complexity of the procedure for processing one negative instance can be derived by duality. It is $O(t_p^{\downarrow} + (P+N)\Sigma_p t_c)$.

6.5.2.6 The Algorithm of the Operation *Collapsed?*

The worst-case time complexity of the IBBS algorithm of the operation *Collapsed?* is $O((1+t^{\uparrow}+P\Gamma t_c+1+N)+(1+t^{\downarrow}+N\Sigma t_c+1+P))$ (see figure 6.9). The part $O(1+t^{\uparrow}+P\Gamma+1+N)$ is due to the IPL collapsed algorithm (see figure 6.7). The first term $O(1)$ arises because we have to check whether the set I^- is empty. The term $O(t^{\uparrow} + P\Gamma t_c)$ arises because it is the time complexity of forming the maximal boundary set G. (The sub-term $O(t^{\uparrow})$ is the worst-case time complexity for generating the set $MAX(Lc)$; the sub-term $O(P\Gamma t_c)$ is the worst-case time complexity of removing those elements of the set $MAX(Lc)$ that do not cover at least one instance of the set I^+.) The second term $O(1)$ arises because we have to check whether the set G is empty. The term $O(N)$ arises because each maximal boundary set $G(n)$ is checked for collapse. The part $O(1+t^{\downarrow}+N\Sigma t_c+1+P)$ is due to the UPL collapsed algorithm (see figure 6.8). Its terms can be explained by analogy. Thus, the overall worst-case time complexity of the IBBS algorithm of the operation *Collapsed?* is $O((1+t^{\uparrow}+P\Gamma t_c+1+N)+(1+t^{\downarrow}+N\Sigma t_c+1+P)) = O(t^{\downarrow}+t^{\uparrow}+(P\Gamma+N\Sigma)t_c)$.

6.5.2.7 The Algorithm of the Operation *Classify*

The IBBS algorithm of the operation *Classify* consists of two procedures for classifying instances as positive and negative, respectively. Thus, the time-complexity analysis of the algorithm is realised for each of the procedures separately.

The worst-case time complexity of the procedure for positive classification of an instance is equal to the worst case time complexity $O(t_n^{\uparrow} + P(\Sigma_p + \Gamma_n)t_c)$ of the operation *Update* (given a negative instance) plus the worst-time complexity $O(t^{\downarrow} + t^{\uparrow} + (P\Gamma + N\Sigma)t_c)$ of the operation *Collapsed?* (see figure 6.10). Thus, the worst-case time complexity of the positive classification of an instance is $O(t_n^{\uparrow} + P(\Sigma_p+\Gamma_n)t_c+t^{\downarrow}+t^{\uparrow}+(P\Gamma+N\Sigma)t_c) = O(t^{\downarrow}+t^{\uparrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+P(\Sigma_p+\Gamma_n)t_c)$.

The worst-case time complexity of the procedure for negative classification of an instance can be derived by duality. It is equal to $O(t^{\downarrow} + t^{\uparrow} + t_p^{\downarrow} + (P\Gamma + N\Sigma)t_c + N(\Sigma_p + \Gamma_n)t_c)$.

The worst-case time complexity of the algorithm of the operation *Classify* is equal to the sum of the worst-case time complexity of the algorithm of the operation

Collapsed?, the worst-case time complexity of the positive classification, and the worst-case time complexity of the negative classification (see figure 6.10). Thus, it is equal to $O((t^{\downarrow}+t^{\uparrow}+(P\Gamma+N\Sigma)t_c)+(t^{\downarrow}+t^{\uparrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+P(\Sigma_p+\Gamma_n)t_c)+(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+(P\Gamma+N\Sigma)t_c+N(\Sigma_p+\Gamma_n)t_c))=O(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+(P+N)(\Sigma_p+\Gamma_n)t_c)$.

6.5.2.8 The Algorithm of the Operation *Member?*

The worst-case time complexity of the IBBS algorithm of the operation *Member?* is $O(P\Sigma_p t_c + N\Gamma_n t_c)$ (see figure 6.11). The term $O(P\Sigma_p t_c)$ arises because all the elements of P minimal boundary sets $S(p)$ with size Σ_p have to be checked whether they are more specific than a given concept description c. The term $O(P\Gamma_n t_c)$ arises because all the elements of N maximal boundary sets $G(n)$ with size Γ_n have to be checked whether they are more general than c.

6.5.2.9 The Algorithm of the Operation *Intersection*

The worst-case time complexity of the IBBS algorithm of the operation *Intersection* is equal to the sum of the time complexities for forming the sets $S_{12}(p)$ and $G_{12}(n)$ (see figure 6.12). To form a set $S_{12}(p)$ we visit Σ_p elements of a set $S_1(p)$ or a set $S_2(p)$. Each element is tested not to cover N instances of the set $I_2^- - I_1^-$ or the set $I_1^- - I_2^-$. Thus, forming each set $S_{12}(p)$ takes $O(N\Sigma_p t_c)$, and generating the sets $S_{12}(p)$ for all $p \in I_1^+ \cup I_2^+$ takes $O(PN\Sigma_p t_c)$. By duality, generating the sets $G_{12}(n)$ for all $n \in I_1^- \cup I_2^-$ takes $O(PN\Gamma_n t_c)$. Thus, the worst-case complexity of the intersection algorithm is $O(PN\Sigma_p t_c + PN\Gamma_n t_c) = O(PN(\Sigma_p+\Gamma_n)t_c)$.

6.5.2.10 The Algorithm of the Operation *Subset?*

The worst-case time complexity of the IBBS algorithm of the operation *Subset?* is equal to P times the worst-case time complexity of the algorithm of the operation *Classify* (positive classification) plus N times the worst-case time complexity of the algorithm of the operation *Classify* (negative classification) (see figure 6.13). Thus, it is equal to $O(P(t^{\downarrow}+t^{\uparrow}+t_n^{\uparrow}+(P\Gamma+N\Sigma)t_c+P(\Sigma_p+\Gamma_n)t_c)+N(t^{\downarrow}+t^{\uparrow}+t_p^{\downarrow}+(P\Gamma+N\Sigma)t_c+N(\Sigma_p+\Gamma_n)t_c))$.

6.5.2.11 The Algorithm of the Operation *Equal?*

The worst-case time complexity of the IBBS algorithm of the operation *Equal?* is two times the worst-case time complexity of the algorithm of the operation *Subset?* (see figure 6.14). So, both are of the same order.

6.6 Adequacy of the Representation

In this section we investigate the tractability and the epistemological and heuristical adequacy of the IBBS representation for the basic version-space operations.

6.6.1 Epistemological Adequacy for the Basic Version-Space Operations

In section 6.4 we have presented the IBBS algorithms of a subset of the basic version-space operations including the operations *Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset?* and *Equal?*. The correctness of the algorithms is proven at least for IPL and UPL languages (see table 6.2). Hence, the IBBS representation is epistemologically adequate with respect to this subset of operations for the classes of IPL and UPL concept languages[4].

Algorithms	Class of Concept Languages
Initialise	admissible
Update	admissible
Retraction	admissible
$Retraction^E$	admissible
Collapsed?	IPL and UPL
Classify	IPL and UPL
Member?	admissible
Intersection	admissible
Subset?	IPL and UPL
Equal?	IPL and UPL

Table 6.2: The Classes of Concept Languages for which the Correctness of the IBBS Algorithms of the Basic Version-Space Operations is Proven.

6.6.2 Heuristical Adequacy for the Basic Version-Space Operations

We investigate the heuristical adequacy of the IBBS with respect to the basic version-space operations *Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset?* and *Equal?* for the classes of IPL and UPL languages. We use the results of the worst-case time-complexity analysis from subsection 6.5.2. They show that the time complexities of the IBBS algorithms of the version-space operations are polynomial in the numbers P and N of training instances, the sizes

[4] This is due to the fact that the IPL and UPL languages are subclasses of the admissible languages.

Σ, Γ, Σ_p and Γ_n, and the complexities $t^{\downarrow}$, $t^{\uparrow}$, $t_p^{\downarrow}$, $t_n^{\uparrow}$, and t_c. Note that each of these sizes and complexities can depend only on the concept languages used. Thus, all the algorithms are tractable; i.e., *IBBS are heuristically adequate for the basic version-space operations Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset? and Equal? for the classes of IPL and UPL concept languages, if the sizes* Σ, Γ, Σ_p *and* Γ_n, *and the complexities* $t^{\downarrow}$, $t^{\uparrow}$, $t_p^{\downarrow}$, $t_n^{\uparrow}$, *and* t_c *are polynomial in relevant properties of these languages.*

6.6.3 Tractability

As discussed earlier (see section 5.1), the IBBS are tractable when they are heuristically adequate for the operations *Initialise* and *Update*, and their size is polynomial in the number of training instances and relevant properties of the concept languages used. We consider these two conditions in detail.

In subsections 6.5.2.2 and 6.5.2.3 we have established that the sizes Σ and Γ as well as the generation complexities $t^{\downarrow}$ and $t^{\uparrow}$ are not presented in the analytical expressions of the complexities of the operations *Initialise* and *Update*. Thus, in contrast with the general result of the previous section, the IBBS are heuristically adequate for the operations *Initialise* and *Update* for IPL and UPL languages, if only the sizes Σ_p and Γ_n, and the complexities $t_p^{\downarrow}$, $t_n^{\uparrow}$, and t_c are polynomial in relevant properties of these languages.

In section 6.5.1 we have derived the size $O(P\Sigma_p + N\Gamma_n)$ of the IBBS for the worst case. Thus, *IBBS are tractable for the classes of IPL and UPL languages when they are heuristically adequate for the operations Initialise and Update.*

6.6.4 Example

Consider as example a problem for which the instance language Li and the concept language Lc are 1-CNF languages with M Boolean attributes and the single representation trick holds; i.e., $Li \subseteq Lc$ (Cohen and Feigenbaum, 1981). It is easy to prove that 1-CNF languages with Boolean attributes are a subclass of IPL concept languages. Hence, the IBBS are epistemologically adequate with respect to the basic version-space operations *Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset?* and *Equal?* for 1-CNF languages with Boolean attributes. Therefore, we can make a complexity analysis of the IBBS and the algorithms of the operations. We start with the sizes Σ, Γ, Σ_p and Γ_n, and the complexities $t^{\downarrow}$, $t^{\uparrow}$, $t_p^{\downarrow}$, $t_n^{\uparrow}$, and t_c. For 1-CNF languages with Boolean attributes they are as follows: $\Sigma = 1$, $\Gamma = 1$, $\Sigma_p = 1$, $\Gamma_n = M$, $t^{\downarrow} = O(1)$, $t^{\uparrow} = O(1)$, $t_p^{\downarrow} = O(1)$, $t_n^{\uparrow} = O(M)$, and $t_c = M$. Thus, from the space-complexity analysis in subsection 6.5.1 we can derive the worst-case space complexity of the IBBS that is $O(NM)$. Analogously, from the time-complexity analysis from subsection 6.5.2 we can derive the worst-case time complexities of the algorithms of the operations given in table 6.3.

Analysing the worst-case complexities in table 6.3 we conclude that:

Algorithms	Time Complexity
Initialise	$O(1)$
Update ($\oplus$ instance)	$O(NM^2)$
Update ($\ominus$ instance)	$O(PM^2)$
Retraction ($\oplus$ instance)	$O(PNM^2)$
Retraction ($\ominus$ instance)	$O(PNM)$
Retraction^E ($\oplus$ instance)	$O((P+N)M^2)$
Retraction^E ($\ominus$ instance)	$O((P+N)M)$
Collapsed?	$O((P+N)M)$
Classify ($\oplus$ instance)	$O(PM^2)$
Classify ($\ominus$ instance)	$O(NM^2)$
Classify	$O((P+N)M^2)$
Member?	$O(NM^2)$
Intersection	$O(PNM)$
Subset?	$O(P^2M^2+N^2M^2)$
Equal?	$O(P^2M^2+N^2M^2)$

Table 6.3: Worst-Case Time Complexities of the IBBS Algorithms of the Basic Version-Space Operations for 1-CNF Languages with Boolean Attributes.

- the IBBS are *heuristically* adequate with respect to the basic version-space operations, presented in the table, for 1-CNF languages with Boolean attributes;
- the IBBS are tractable for 1-CNF languages with Boolean attributes.

6.7 Experiments

The IBBS algorithms of the version-space operations have been implemented for 1-CNF languages with discrete attributes[5] (using as programming language Java2). We analyse the results of an experimental study of the algorithms for a concept-learning task given in Appendix C. The experiments have been run on a computer with a Pentium II-333MHz processor and 128MB RAM.

The concept-learning task has a concept language Lc with 32 binary attributes that is a super set of the instance language Li (i.e., the single representation trick holds). The training sets I^+ and I^- have 8 instances each. The training instances are chosen so that the size of the maximal boundary set of the version space of the task is $2^{\frac{n}{4}} = 256$ where $n = 32$; i.e., there is an exponential dependency between the

[5] Note that the class of 1-CNF languages with discrete attributes is a subclass of the class of intersection-preserving languages.

Figure 6.15: The Average Size of IBBS.

size of the maximal boundary set and the number of the negative training instances (following Haussler (1988) worst case).

The size of the IBBS is measured by the sum of all their minimal and maximal elements. The average size of IBBS is given in figure 6.15. It has a local maximum between the first and fourth training instances. This reflects the situation when the size of the set I^+ of positive instances is much smaller than the size of the set I^- of negative instances. In this case the G-part of the IBBS grows almost unrestricted by positive instances. When more positive instances become available (see the graph after the second instance) the size of the IBBS decreases and after the fourth instance it grows linearly in the number of training instances.

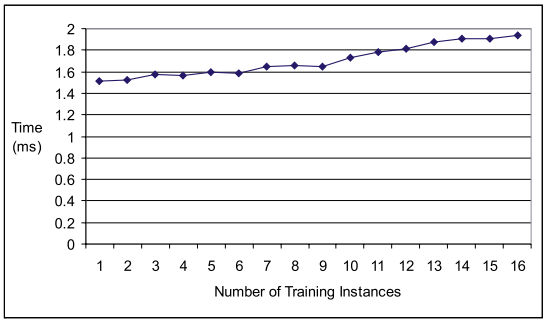

Figure 6.16: The Average Running Time of the Algorithm of the Operation *Update*.

The average running time of the algorithm of the operation *Update* is given in figure 6.16. It is approximately linear in the number of training instances and it does not have a local maximum for small training sizes. This contrasts with the size of the IBBS and can be explained for two opposite cases:

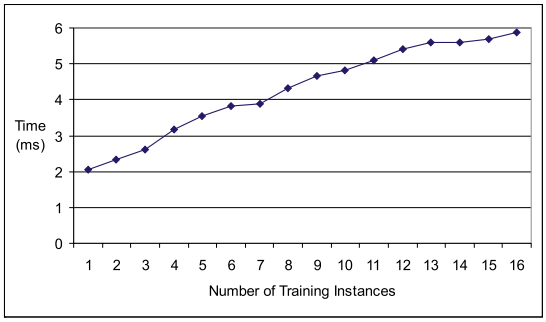

Figure 6.17: The Average Running Time of the Efficient Algorithm of the Operation *Retraction*.

- $I^+ = \emptyset$ and $I^- \neq \emptyset$. In this case the update algorithm generates only the maximal boundary sets $G(n)$ corresponding to the current negative instances n and adds them to the IBBS. Thus, no extra computations are required.
- $I^+ \neq \emptyset$ and $I^- = \emptyset$. In this case the update algorithm generates only the minimal boundary sets $S(p)$ corresponding to the current positive instances p and adds them to the IBBS. Thus, no extra computations are required.

Since constraints 6.23 and 6.24 hold for 1-CNF languages with Boolean attributes we have implemented the efficient algorithm of the operation *Retraction*. The average running time of the algorithm is given in figure 6.17. It is approximately linear in the number of training instances. It is higher than that of the update algorithm which is in accordance with the worst-case time-complexity analysis (see table 6.3).

Note that the average running time of the algorithm of the operation *Retraction* does not have a local maximum for small training sizes. In this respect the algorithm is similar to the update algorithm and it can be explained for two opposite cases:

- $I^+ = \emptyset$ and $I^- \neq \emptyset$. In this case the retraction algorithm finds and removes only the maximal boundary sets $G(n)$ corresponding to the current negative instances n to be retracted. Thus, no extra computations are required.
- $I^+ \neq \emptyset$ and $I^- = \emptyset$. In this case the retraction algorithm finds and removes only the minimal boundary sets $S(p)$ corresponding to the current positive instances p to be retracted. Thus, no extra computations are required.

Since 1-CNF languages with discrete attributes are a subclass of IPL languages, we have implemented the IPL version of the algorithm of operation *Collapsed?*. The average running time of the algorithm is given in figure 6.18. It has a local maximum of 0.58 milliseconds for the first training instance so that for the next

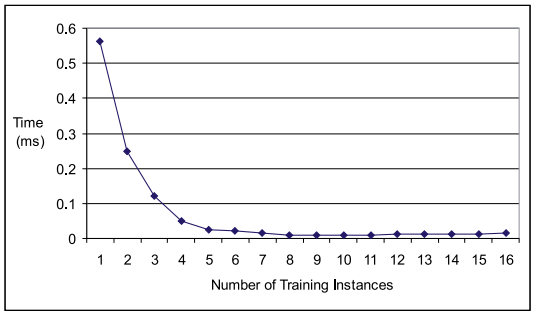

Figure 6.18: The Average Running Time of the Algorithm of the Operation *Collapsed?*.

seven instances it drops to 0.012 milliseconds. After the eight instance the running time is approximately linear in the number of training instances. The pick at the beginning reflects the situation when the set I^- of negative instances is empty. In this case the algorithm has to form the set $MAX(Lc)$ of maximal descriptions in the concept language Lc used, and to check whether each element of the set covers all the positive instances. Thus, we can conclude that the algorithm of the operation *Collapsed?* is efficient for IPL languages when the set I^- of negative instances is not empty.

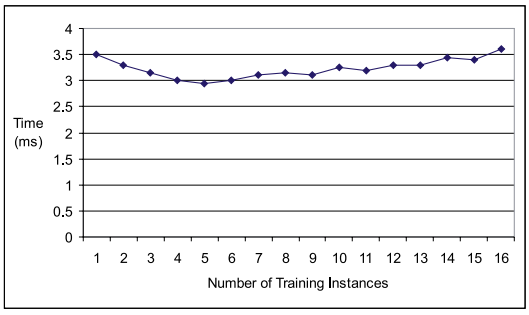

Figure 6.19: The Average Running Time of the Algorithm of the Operation *Classify*.

The average running time of the algorithm of the operation *Classify* is given in figure 6.19. It has a local maximum for the first training instance. For the next four instances the running time decreases. This is due to the fact that the algorithm uses the operation *Collapsed?* which algorithm has this property. After the sixth instance the running time becomes approximately linear in the number of training instances. This is due to the fact that the algorithm applies twice the update algorithm in the

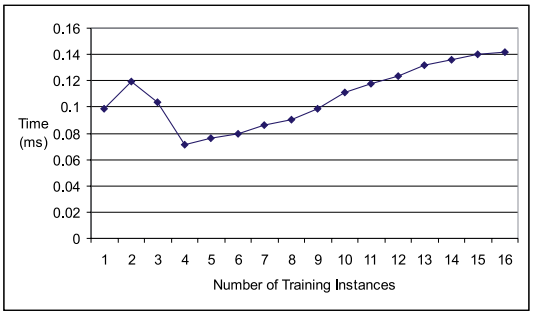

Figure 6.20: The Average Running Time of the Algorithm of the Operation *Member?*.

worst case. Thus, the time is higher than that of the update algorithm but still less than the time of the retraction algorithm.

The average running time of the algorithm of the operation *Member?* is given in figure 6.20. It has a local maximum between the first and fourth training instance. This is due to the same property of the size of the IBBS. After the fourth instance the running time becomes approximately linear in the number of training instances and is much less than that of the update algorithm. We explain this observation with high complexities for generating the boundary sets of version spaces that increases the running time of the update algorithm.

Figure 6.21: The Average Running Time of the Algorithm of the Operation *Intersection*.

The average running time of the algorithm of the operation *Intersection?* is studied for the case when the version space to be learned is intersected with itself. It is given in figure 6.21. The graph of the figure has a local maximum between the first and fourth training instance. This is due to the same property of the size of the IBBS. After the fourth instance the running time becomes approximately linear

Figure 6.22: The Average Running Time of the Algorithm of the Operation *Subset?*.

in the number of training instances and it is higher than the time of the retraction algorithm.

The average running time of the algorithm of the operation *Subset?* is studied for the case when the version space to be learned is compared with itself. It is given in figure 6.22. The graph of the figure shows that the running time is approximately linear in the number of training instances and it is higher than the time of the intersection algorithm. This result is confirmed by the complexity analysis.

Figure 6.23: The Average Running Time of the Algorithm of the Operation *Equal?*.

The average running time of the algorithm of the operation *Equal?* is studied for the case when the version space to be learned is compared to itself. It is given in figure 6.23 and it is approximately two times the time of the subset algorithm.

6.8 Comparison with Related Work

The IBBS representation can be compared with the existing version-space representations. We start the comparison with the landmark work of Haussler (1988). The work presents an example of the boundary sets which shows that their size can grow exponentially in the number of training instances for 1-CNF languages with Boolean attributes. (For the sake of completeness we have shown the example in section 5.4.5.) Analysing the example, we have found that its expression of the maximal boundary set is essentially the IBBS representation of the set. Unfortunately, Haussler never considered the example in the context of version-space representations. That is why we have filled this omission by a complete development of the IBBS representation.

We continue our comparison with the list representation of version spaces (Mitchell, 1978; Hirsh, 1992b). The representation is less efficient than the IBBS since (1) it is epistemologically adequate with respect to all the basic version-space operations for finite concept languages; and (2) it is tractable and heuristically adequate with respect to all the basic version-space operations for small concept languages (see section 5.2).

The IBBS representation is closely related to the boundary sets (Mitchell, 1978). The main advantage of the boundary sets is that they are epistemologically adequate with respect to the whole set of basic version-space operations for the class of admissible languages while the IBBS representation is epistemologically adequate only with respect to a strict subset of these operations for the classes of IPL and UPL languages. The strong disadvantages of the boundary sets are the computational problem and the retraction problem that we have formulated in our problem statement. The IBBS overcome both problems for the classes of IPL and UPL concept languages when the conditions for tractability and heuristical adequacy of the representation with respect to the basic version-space operations hold.

We continue the comparison with four alternative version-space representations (see chapter 1). The first alternative version-space representation was proposed by Idemstam-Almquist (1990). The representation was designed especially for the operation *Retraction* when concept languages are 1-CNF languages with tree-structured attributes. The representation consists of the tree-structured domains of all the attributes plus special markers that reflect the influence of each training instance on the domains. When a new training instance is given the algorithm of the operation *Update* adds the instance markers to the representation. Conversely, when a processed training instance is retracted the algorithm of the operation *Retraction* removes the instance markers from the representation. Both algorithms are simple and it is possible even to prove that they are tractable together with the representation itself for 1-CNF languages with tree-structured attributes. Unfortunately, the representation is language-dependent and that is why it was not generalised for broader classes of concept languages. This contrasts with the IBBS that can be

applied for the classes of IPL and UPL concept languages.

The next alternative version-space representation was proposed by Smith and Rosenbloom (1990) for 1-CNF languages with tree-structured attributes as well. The representation is essentially boundary sets. Its minimal boundary set contains only one element since the languages used are 1-CNF with tree-structured attributes (see theorem 6.11). When a positive instance is given the algorithm of the operation *Update* coincides with that of the boundary sets. When a negative instance is given the algorithm of the operation *Update* revises the boundary sets if the instance is near-miss with respect to the only minimal boundary element. Otherwise, the instance is memorised. An instance is a near-miss if it is not covered by only one attribute of the minimal boundary element (Winston, 1992). It was proven that the near-miss negative instances do not fragment the maximal boundary set. Thus, the update algorithm can be considered as a partial solution to the computational problem of the boundary sets since it requires negative near-miss instances. This contrasts with the IBBS because they do not impose any constraints on the type of the training instances. In addition to that the representation of Smith and Rosenbloom is not efficient for the operation *Retraction* since it is essentially boundary sets.

Hirsh (1992b) continued the main line of the previous two works in his unilateral boundary sets. The main advantage of unilateral boundary sets is that they are epistemologically adequate for the set of all the basic version-space operations for the class of either lower-admissible or upper-admissible languages while the IBBS representation is epistemologically adequate for a subset of these operations for the classes of IPL and UPL languages. The common properties of both representations are that they can be applied when either the minimal or the maximal boundary sets grow exponentially. The disadvantage of the unilateral boundary sets is that we have to know in advance which boundary set grows exponentially. This is not a problem with the IBBS representation since both boundaries of version spaces are instance-based. The second disadvantage of unilateral boundary sets is that they do not have an efficient algorithm of the operation *Retraction*, while the IBBS representation has (see subsection 6.4.1.5).

Hirsh *et al.* (1997) generalised the progress in alternative version-space representations. In addition to that they proposed a new version-space representation. The key idea is that the algorithms of almost all basic version-space operations exist when there exists an algorithm for the consistency problem. The consistency problem is to determine the existence of a concept description that correctly classifies training instances. Therefore, the training instances themselves were proposed as a version-space representation. Since the consistency algorithm can be constructed by the operations *Update* and *Collapsed?* we have a consistency algorithm based on the IBBS representation. Thus, all the properties of the IBBS can be inherited by the representation with the training instances.

6.9 Chapter Conclusion

The main contribution of this chapter is the complete development of the IBBS representation of version spaces. The epistemological adequacy of the IBBS has been proven with respect to the basic version-space operations *Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset?* and *Equal?* for the classes of IPL and UPL languages. The conditions for tractability and heuristical adequacy of the representation for the operations have been derived. An analysis of the conditions shows that they can easily be met in practice. Therefore, if the concept languages are either IPL or UPL, we conclude that:

(1) the IBBS can overcome the computational and retraction problems of the boundary sets. This means that the IBBS are a version-space representation that can simultaneously solve the second and third part of our problem statement;

(2) the IBBS together with their algorithms of the basic version-space operations can be used for an efficient implementation of the abstract data type of version spaces.

Using the results of chapter 4 the conclusions (1) and (2) imply that the IBBS can be used for efficient implementations of the abstract data types of conjunctive and disjunctive version spaces. This research direction is considered in the next chapter.

Chapter 7

Implementing Conjunctive and Disjunctive Version-Space Abstract Data Types

This chapter considers the problem of implementing the conjunctive and disjunctive version-space abstract data types. In sections 7.1 and 7.2 we show that the list representation, the boundary sets, the unilateral boundary sets, and the instance-based boundary sets can represent conjunctive and disjunctive version spaces. Moreover, we demonstrate that each of these representations is epistemologically adequate with respect to the basic conjunctive and disjunctive version-space operations for the concept languages for which it has been designed. Thus, we conclude that the conjunctive and disjunctive version-space abstract data types can be implemented using either list representation, boundary sets, unilateral boundary sets, or instance-based boundary sets. For this reason we determine the conditions when these representations are heuristically adequate for the basic conjunctive and disjunctive version-space operations as well as the conditions when they tractably represent conjunctive and disjunctive version spaces.

The chapter ends in section 7.3 where the results presented in the chapter are summarised and the main research line of the next chapter is determined.

7.1 List Representation, Boundary Sets, Unilateral Boundary Sets

In this section we consider the problem of implementing the conjunctive version-space abstract data type using list representation, boundary sets, or unilateral

boundary sets. We propose to solve the problem by combining the results of chapters 4 and 5.

In chapter 4 we have shown that the conjunctive version-space abstract data type can be implemented via the version-space abstract data type. More precisely, we have proven that:

(1) every conjunctive version space can be represented in concept languages by a positive version space and version spaces with respect to negative training instances;

(2) every conjunctive version-space operation can be represented by an algorithm which main operators are the basic version-space operations.

In chapter 5 we have shown that the version-space abstract data type can be implemented using either list representation, boundary sets, or unilateral boundary sets. More precisely, we have proven that:

(3) the representations are correct; i.e., they correctly represent version spaces in concept languages;

(4) the representations are epistemologically adequate with respect to basic version-space operations for the concept languages for which they have been designed.

Combining (1) and (3) it follows that using each of the representations we can represent the positive version space and the version spaces with respect to negative instances of every conjunctive version space. Thus, we conclude that:

(5) conjunctive version spaces can be represented using either list representation, boundary sets, or unilateral boundary sets.

Combining (2) and (4) it follows that:

(6) the list representation, the boundary sets, and the unilateral boundary sets are epistemologically adequate for the basic conjunctive version-space operations. The adequacy holds for the concept languages for which the representations have been designed.

From (5) and (6) we can conclude that *the conjunctive version-space abstract data type can be implemented using either list representation, boundary sets, or unilateral boundary sets.*

To determine the heuristical adequacy of the list representation, the boundary sets, and the unilateral boundary sets for the basic conjunctive version-space operations we consider table 7.1. The table shows the numbers of the basic version-space operations that are necessary for execution of every basic conjunctive version-space

Conjunctive Version Space Operations	Numbers of Version Space Operations
$Initialise_{CVS}$	*Initialise*
$Update_{CVS}$	$(N+1)$ *Update*
$Retraction_{CVS}$	$(N+1)$ *Retraction*
$Collapsed?_{CVS}$	$(N+1)$ *Collapsed?*
$Classify_{CVS}$	$(N+1)$(*Collapsed?* + *Classify*)
$Member?_{CVS}$	$(N+1)\|C\|$ *Member?*
$Intersection_{CVS}$	$(2N+1)$ *Intersection*
$Subset?_{CVS}$	(N^2+1) *Subset?*
$Equal?_{CVS}$	$2N^2$ *Subset?* + *Equal?*

Table 7.1: The Numbers of Basic Version-Space Operations Needed for Execution of Basic Conjunctive Version-Space Operations in the Worst Case (N is the number of negative training instances; $|C|$ is the length of a conjunction).

operation in the worst case. An analysis of the table shows that the algorithms of the basic conjunctive version-space operations are tractable when the algorithms of the basic version-space operations are tractable. Therefore, we can conclude that *given a concept language the list representation, the boundary sets, and the unilateral boundary sets are heuristically adequate for all the basic conjunctive version-space operations when they are heuristically adequate for the basic version-space operations.*

Tractability of a representation of conjunctive version spaces is implied by tractability of that representation when it characterises positive version spaces and version spaces with respect to negative instances[1]. Thus, *given a concept language the list representation, the boundary sets, and the unilateral boundary sets tractably represent conjunctive version spaces when they tractably represent ordinary version spaces.*

Note that in chapter 4 we have determined that conjunctive version spaces are solutions to the problem of incomplete concept languages when the target concepts are represented in conjunctive extension of concept languages. Therefore, we can conclude that the conjunctive version spaces represented by list representation, boundary sets, or unilateral boundary sets provide more detailed solutions to this problem. In addition to that we have shown in chapter 5 that the unilateral boundary sets are a solution to the computational problem of version spaces when the conditions for tractability of the representation hold. Therefore, we can conclude that conjunctive version spaces represented by unilateral boundary sets, provides a simultaneous solution to the problems above that correspond to the first and second part of our problem statement.

[1] This follows from the fact that the number of the version spaces with respect to negative instances is finite since the training sets are finite according to constraint 2.21.

By duality the same derivations can be done for disjunctive version spaces. We do not write out these derivations since they are analogous.

7.2 Instance-Based Boundary Sets

In this section we consider the problem of implementing the conjunctive version-space abstract data type using the instance-based boundary sets (IBBS). We propose to solve the problem by combining the results of chapters 4 and 6.

In chapter 6 we have shown that the IBBS are epistemologically adequate for the basic version-space operations (with the exception of the operation *Converged?*) for intersection-preserving and union-preserving languages (IPL, UPL). Therefore, we determine the relevance of IPL and UPL languages with respect to their conjunctive extensions and conjunctive version spaces.

According to theorem 4.7 if a concept, extensionally represented by a set I in an instance language Li, is intensionally represented in a concept language Lc, that is either IPL or UPL, then the concept is intensionally represented in the conjunctive extension CLc of Lc. An interesting property of the IPL languages is that the opposite implication also holds: if a concept, extensionally represented by a set I in an instance language Li, is intensionally represented in the conjunctive extension CLc of an IPL concept language Lc, then the concept is intensionally represented in Lc. This property of IPL concept languages is proven in theorem 7.1.

Theorem 7.1. *Consider an instance language Li, an IPL concept language Lc, and the conjunctive extension CLc of Lc. Then:*

$$(\forall I \subseteq Li)((\exists c \in Lc)cons(c, \langle I, Li - I\rangle) \leftarrow (\exists C \in CLc)cons_C(C, \langle I, Li - I\rangle)).$$

Proof. Consider an arbitrarily chosen set $I \subseteq Li$ and a conjunction $C \in CLc$ such that $cons_C(C, \langle I, Li - I\rangle)$. From $C \in CLc$ according to definition 4.3 we have that $C \neq \emptyset$ and $C \subseteq Lc$. Using definition 6.9 since Lc is IPL, the last two formulas imply:

$$glb(C) \in Lc \tag{7.1}$$

According to definition 4.5 $cons_C(C, \langle I, Li - I\rangle)$ is equivalent to:

$$(\forall i \in I)M_C(C, i)$$
$$(\forall i \in Li - I)\neg M_C(C, i).$$

Using definition 4.4 the last two formulas imply:

$$(\forall i \in I)(\forall c \in C)M(c, i)$$
$$(\forall i \in Li - I)(\exists c \in C)\neg M(c, i).$$

Thus, since Lc is IPL according to definition 6.9 we have that:

$$(\forall i \in I)M(glb(C), i)$$
$$(\forall i \in Li - I)\neg M(glb(C), i).$$

Using definition 2.22 the last two formulas are equivalent to:

$$cons(glb(C), \langle I, Li - I\rangle). \tag{7.2}$$

Thus, from (7.1) and (7.2) it follows that:

$$(\exists c \in Lc)cons(c, \langle I, Li - I\rangle).$$

□

In contrast to IPL languages the opposite implication cannot be proven for UPL languages. Thus, we conclude that:

(1) the IPL languages have the same level of completeness as their conjunctive extensions. Therefore, it is possible to show that version spaces in IPL languages and conjunctive version spaces in conjunctive extensions of IPL languages have the same level of completeness as well.

(2) the UPL languages are less complete than their conjunctive extensions. Therefore, by corollary 4.11 version spaces in UPL languages are less complete than conjunctive version spaces in conjunctive extensions of UPL languages.

From (1) and (2) it follows that:

(3) if the concept languages are IPL, then the conjunctive version spaces do not provide any solution to the problem of incomplete concept languages;

(4) if the concept languages are UPL, then the conjunctive version spaces do provide a solution to the problem of incomplete concept languages.

Taking into account (3) and (4) *we assume in the rest of the thesis that the conjunctive version spaces are defined in conjunctive extensions of UPL languages only.* In this context we consider the problem of implementing the conjunctive version-space abstract data type by IBBS. We follow the same steps, taken in the previous chapter, in order to conclude that:

(5) the conjunctive version spaces can be represented by IBBS. For example, consider an arbitrary conjunctive version space CVS. By theorem 4.19 CVS is represented by a positive version space $VS(\emptyset)$ and version spaces $VS(n)$ with respect to negative instances. Each of these version space can be represented by IBBS following theorem 6.8. Thus, according to definition 6.5 of the IBBS:

– the positive version space $VS(\emptyset) = \{c \in Lc | (\forall p \in I^+) M(c,p)\}$ is represented by IBBS:$\langle \{S(\{p\},\emptyset)\}_{p \in I^+}, \emptyset \rangle$, where[2]:

$$S(\{p\},\emptyset) = MIN(VS(\{p\},\emptyset))$$
$$VS(\{p\},\emptyset) = \{c \in Lc | M(c,p)\}.$$

– each of the version spaces $VS(n) = \{c \in Lc | cons(c, \langle I^+, \{n\}\rangle)\}$ is represented by IBBS:$\langle \{S(p,\{n\})\}_{p \in I^+}, \{G(n)\}\rangle$, where:

$$S(p,\{n\}) = MIN(VS(p,\{n\}))$$
$$VS(p,\{n\}) = \{c \in Lc | cons(c, \langle \{p\}, \{n\}\rangle)\}$$
$$G(n) = MAX(VS(n)).$$

(6) the IBBS are epistemologically adequate with respect to the basic conjunctive version-space operations for UPL languages.

From (5) and (6) it follows that *the conjunctive version-space abstract data type can be implemented by IBBS.*

Using similar arguments from the previous section we can determine the conditions of heuristical adequacy of the IBBS for the basic conjunctive version-space operations as well as the conditions when they tractably represent conjunctive version spaces. Given a UPL concept language the conditions are as follows:

- *the IBBS are heuristically adequate for all the basic conjunctive version-space operations when they are heuristically adequate for the basic version-space operations*;

- *the IBBS tractably represent conjunctive version spaces when they tractably represent (ordinary) version spaces.*

In the previous section we have repeated the main contribution of chapter 4, namely that the conjunctive version spaces are a partial solution to the problem of incomplete concept languages. Therefore, we can conclude that the conjunctive version spaces represented by IBBS provides a more detailed solution to this problem. In addition to that we have shown in chapter 6 that the IBBS are a solution to the computational and retraction problems of version spaces when the conditions for heuristical adequacy and tractability of the representation hold. Therefore, we can conclude that the conjunctive version spaces represented by IBBS can solve simultaneously the problems above.

By duality the same derivations can be done for IPL languages and disjunctive version spaces. We do not work out this part of the section since it is analogous.

[2] Note that we use here the complete notation 4.14.

7.3 Chapter Conclusion

The main contribution of this chapter is that we have shown that the conjunctive and disjunctive version-space abstract data types can be implemented using either list representation, boundary sets, unilateral boundary sets, or instance-based boundary sets. For that reason we have analysed the adequacy and the tractability of these representations for conjunctive and disjunctive version spaces and their basic operations. In this context *the conjunctive and disjunctive version spaces represented by instance-based boundary sets have been considered as a solution to the problem with incomplete concept languages, the computational problem, and the retraction problem, that correspond to the first, second, and third part of our problem statement.*

The problem with noisy training instances, that constitutes the last, fourth part of the problem statement, is proposed to be solved in the next chapter. The solution is given for conjunctive and disjunctive version-spaces represented by instance-based boundary sets.

Chapter 8

Instance-Based Classification

This chapter answers the fourth part of our problem statement. It proposes a solution to the problem of noisy training instances for conjunctive and disjunctive version spaces represented by instance-based boundary sets. The solution is based on classification methods taken from instance-based learning (Duda and Hart, 1973; Dasarathy, 1991; Aha, Kibler, and Albert, 1991; Aha, 1997). Therefore, we start the chapter in section 8.1 where the task of instance-based learning is specified and its basic (k-nearest neighbour) algorithm is described.

The proposed solution for the conjunctive version spaces is considered in section 8.2. The key idea is to apply the k-nearest-neighbour algorithm to the S-part and G-part of the instance-based boundary sets of the conjunctive version spaces. This results in two instance-based classification schemes S and G introduced in subsections 8.2.1 and 8.2.2. We show that both schemes implement preference biases. Nevertheless, the schemes do not require any change in the instance-based boundary-set representation of the conjunctive version spaces. Hence, they can be added to the conjunctive version-space abstract data type implemented using instance-based boundary sets.

In subsection 8.2.3 the instance-based classification schemes S and G are generalised for the concept-learning tasks with an arbitrary number of target concepts. We analyse the generalised schemes and emphasise their advantages and disadvantages. We determine the generalised scheme S as optimal since its classification performance does not degrade when the number of target concepts increases.

In subsections 8.2.4 and 8.2.5 the classification schemes S and G are proposed to be combined with the condensed nearest-neighbour algorithm (Hart, 1968), and the edited nearest-neighbour algorithm (Wilson, 1972), respectively. The combinations aim at reducing the computational complexity and the noise sensitivity of the schemes.

In subsections 8.2.6 and 8.2.8 we describe two systems that are based on the generalised classification schemes S and G. The systems have been tested on eight

datasets from the Machine Learning Database Repository at the University of California (Blake and Merz, 1998). The results of experiments are presented in subsection 8.2.7 where they are compared with the results of the C4.5 decision-tree learner (Quinlan, 1993). The comparison shows that the systems and the C4.5 learner have comparable classification performance. The combinations of the systems with the condensed nearest-neighbour algorithm are tested on the same datasets. The results are not encouraging since the desired drop in the computational complexity is accompanied by a significant drop in the classification performance. This contrasts with the combinations of the systems with the edited nearest-neighbour algorithm. They are tested on the same data sets in the presence of 10% random class noise. The results show that the combinations of the systems with the edited nearest-neighbour algorithm are more robust with respect to noise in training data than the systems themselves. Therefore, we conclude that the fourth part of our problem statement is solved, namely the problem with noisy training instances.

In subsection 8.3 we claim that by duality analogous classification schemes can be designed for disjunctive version spaces represented by instance-based boundary sets.

The instance-based classification schemes S and G of conjunctive and disjunctive version spaces are compared with related work on version-space learning. The comparison is made in subsection 8.4 and it is in favour of the schemes.

The chapter ends in section 8.5. The section contains chapter conclusions.

8.1 Instance-Based Learning

Instance-based learning is a sub-field of machine learning (Duda and Hart, 1973; Dasarathy, 1991; Aha *et al.*, 1991; Aha, 1997). Its basic task can be formalised as a quadruple $\langle Li, D, T, \{I^t\}_{t \in T} \rangle$ presented in figure 8.1. Given an instance language Li, a distance function D, a set T of target concepts, training sets I^t of the concepts $t \in T$, the goal of the task is to classify accurately instances in the language Li. Note that the task does not include the concept language and the membership relation. Instead, it has the distance function D that determines the distance between each two instances $i_1, i_2 \in Li$.

The basic instance-based learning algorithm is the k-nearest-neighbour algorithm (Cover and Hart, 1967; Duda and Hart, 1973; Dasarathy, 1991). Given an instance $i \in Li$ to be classified it finds k nearest instances in the set $\bigcup_{t \in T} I^t$ using the distance function D. Then the algorithm classifies the instance to belong to a concept $t \in T$ which instances form a majority among the k nearest instances.

The inductive bias of the k-nearest-neighbour algorithm "corresponds to an assumption that the classification of an instance will be most similar to the classification of other instances that are nearby" according to the distance function D (Mitchell, 1997, p.234). That is why the algorithm is intuitive and easy to understand, which facilitates its implementation and modification.

The Instance-Based Learning Task
$\langle Li, D, \{I^t\}_{t \in T} \rangle$.

Given:

- An instance language Li.
- A distance function D.
- A set T of target concepts.
- Training sets $I^t \subseteq Li$ of target concepts $t \in T$.

Goal: Classify accurately instances in the language Li.

Figure 8.1: The Instance-Based Learning Task.

If we continue to count the advantages of the k-nearest-neighbour algorithm we have to mention that:

(1a) it learns quickly by memorising the training instances;

(2a) it models complex target concepts;

(3a) it provides good classification performance on many application domains (see (Wilson and Martinez, 2000)).

However, in its basic form the k-nearest-neighbour algorithm has several drawbacks:

(1d) it has high space complexity since all the training instances are memorised;

(2d) it has high time complexity since all the training instances have be visited in order to classify each new instance;

(3d) its classification performance degrades with the introduction of noise in the training instances.

In the next two sections we show how to use the k-nearest-neighbour algorithm in the context of conjunctive version spaces represented by instance-based boundary sets. The emphasis will be on the classification process plus ways to overcome the mentioned drawbacks of the algorithm.

8.2 Instance-Based Classification with Conjunctive Version Spaces

We propose instance-based classification with conjunctive version spaces when they are represented by instance-based boundary sets (IBBS). The key idea is simple: the elements of the IBBS correspond to particular training instances. Therefore, we can apply the k-nearest-neighbour algorithm on the IBBS.

We elaborate the key idea in the rest of this chapter when the following three assumptions hold:

(1) concept languages are union preserving (UPL);

(2) conjunctive extensions of concept languages are complete;

(3) conjunctive version spaces are represented by IBBS.

In addition to the three assumptions, for the sake of simplicity of presentation, we assume that only the sets I^+ and I^- of positive and negative training instances of a target concept are given. Using the terminology of instance-based learning the first set I^+ is a set of instances of the target concept, the second set I^- is a set of instances of a concept that is opposite to the target concept[1]. In order to apply instance-based classification schemes we have to make both concepts equipollent. Thus, we consider two conjunctive version spaces: the conjunctive version space CVS^+ of the target concept, and the conjunctive version space CVS^- of the concept that is opposite to the target concept.

By theorem 4.19 a conjunctive version space can be represented by the positive version space and version spaces with respect to negative instances. By theorem 6.8 each of these version spaces can be represented by IBBS for UPL concept languages. Using the complete notation 4.14 we summarise the representations for the conjunctive version spaces CVS^+ and CVS^- in tables 8.1 and 8.2, respectively.

Version Spaces	IBBS
$VS^+(\emptyset) = \{c \in Lc \mid (\forall p \in I^+) M(c,p)\}$	$\langle \{S^+(\{p\},\emptyset)\}_{p \in I^+}, \emptyset \rangle$ where $S^+(\{p\},\emptyset) = MIN(\{c \in Lc \mid M(c,p)\})$
$VS^+(n) = \{c \in Lc \mid cons(c, \langle I^+, \{n\}\rangle)\}$ for $n \in I^-$	$\langle \{S^+(p,\{n\})\}_{p \in I^+}, \{G^+(n)\}\rangle$ where $S^+(p,\{n\}) = MIN(\{c \in Lc \mid cons(c, \langle\{p\},\{n\}\rangle)\})$ $G^+(n) = MAX(VS(n))$

Table 8.1: The Conjunctive Version Space CVS^+: the IBBS of the Positive Version Space $VS^+(\emptyset)$ and the Version Spaces $VS^+(n)$ with respect to Negative Instances.

[1] The concept-learning tasks containing training sets of two target concepts only are known as binary concept-learning tasks. We use this term in the remainder of the thesis.

Note that in the context of the conjunctive version space CVS^- the set I^- is the set of positive training instances and the set I^+ is the set of negative training instances. Therefore, in this case an instance in the set I^- is denoted by p, and an instance in the set I^+ is denoted by n. This allows to preserve the notation 4.14.

Version Spaces	IBBS
$VS^-(\emptyset) = \{c \in Lc \mid (\forall p \in I^-) M(c,p)\}$	$\langle \{S^-(\{p\},\emptyset)\}_{p \in I^-}, \emptyset \rangle$ where $S^-(\{p\},\emptyset) = MIN(\{c \in Lc \mid M(c,p)\})$
$VS^-(n) = \{c \in Lc \mid cons(c, \langle I^-, \{n\}\rangle)\}$ for $n \in I^+$	$\langle \{S^-(p,\{n\})\}_{p \in I^-}, \{G^-(n)\}\rangle$ where $S^-(p,\{n\}) = MIN(\{c \in Lc \mid cons(c, \langle \{p\},\{n\}\rangle)\})$ $G^-(n) = MAX(VS(n))$

Table 8.2: The Conjunctive Version Space CVS^-: the IBBS of the Positive Version Space $VS^-(\emptyset)$ and the Version Spaces $VS^-(n)$ with respect to Negative Instances.

The elements of the IBBS of the conjunctive version spaces CVS^+ and CVS^- correspond to particular training instances. In order to use them in instance-based classification, based on the k-nearest-neighbour algorithm, we specify the distance function D. The function differs from ordinary distance functions. It measures how an instance $i \in Li$ corresponds to a given set $S \subseteq Lc$; i.e., how many elements in S do not cover the instance. More formally:

$$D(S,i) = |\{s \in S \mid \neg M(s,i)\}|.$$

An analysis of tables 8.1 and 8.2 shows that using the distance function D we can classify either with the IBBS of the positive version spaces $VS^+(\emptyset)$ and $VS^-(\emptyset)$, or with the IBBS of the version spaces $VS^+(n)$ and $VS^-(n)$. In the next two subsections we consider these two schemes.

8.2.1 Scheme S

The instance-based classification scheme S is based on the S-part of the IBBS of the positive version spaces $VS^+(\emptyset)$ and $VS^-(\emptyset)$. Given the distance function D it is an adaptation of the k-nearest-neighbour algorithm. When an instance $i \in Li$ has to be classified its k nearest minimal boundary sets $S^+(\{p\},\emptyset)$ and $S^-(\{p\},\emptyset)$ are found according to the distance function $D(S,i)$. Then concept is determined which minimal boundary sets form the majority among the k nearest sets. The found concept is the classification of the instance i.

The classification of every instance, determined by the scheme S, can be explained by a simple algorithm. The algorithm forms sets S_c^+ and S_c^- from those elements of the nearest sets $S^+(\{p\},\emptyset)$ and $S^-(\{p\},\emptyset)$, respectively, that do cover the instance. Since the concept languages are UPL the algorithm generates the least upper bounds

of the sets S_c^+ and S_c^-. The upper bounds are used in order to facilitate the user to understand the resulting classification of any instance.

The inductive bias of the scheme S is a preference bias. This is due to the fact that an instance is classified to belong to a concept which minimal boundary sets are a majority among the k-nearest minimal boundary sets for the instance. Hence, there exists a preference to the nearest minimal boundary sets of the winning concept.

8.2.2 Scheme G

Instance-based classification scheme G is based on the G-part of the IBBS of the version spaces $VS^+(n)$ and $VS^-(n)$. To explain the scheme we trace the following chain of reasoning. We have assumed that (1) the concept languages used are UPL, and (2) their conjunctive extensions are complete. By theorem 6.12 (1) implies that each set $G^+(n)$ has only one element $g^+(n)$. By theorem 4.9 (2) implies that for each $n \in I^-$ there exists a description $c \in Lc$ such that $cons(c, \langle Li - \{n\}, \{n\}\rangle)$. Thus, by corollary 6.14 it is possible to prove that:

$$(\forall n \in I^-) cons(g^+(n), \langle Li - \{n\}, \{n\}\rangle).$$

Since we have found that each set $G^+(n)$ has only one element $g^+(n)$ the last derivation implies:

$$(\forall n_1, n_2 \in I^-)((n_1 \neq n_2) \leftrightarrow (g^+(n_1) \neq g^+(n_2))).$$

Therefore, the instances n are unique for the corresponding maximal elements $g^+(n)$, and the maximal elements $g^+(n)$ are unique for the corresponding instances n.

By analogy the same derivations can be obtained for the maximal boundary sets $G^-(n)$ of the version spaces $VS^-(n)$ with respect to instances $n \in I^+$. Thus, we can conclude that the maximal boundary elements $g^+(n)$ and $g^-(n)$ can not be used for instance-based classification since each of them covers the whole instance language with the exception of the corresponding instance n. To avoid this problem we introduce two types of sets:

- $MING^+(n) = MIN(\{c \in Lc | (\exists i \in Li) M(c,i) \wedge c \leq g^+(n)\})$ for all $n \in I^-$
- $MING^-(n) = MIN(\{c \in Lc | (\exists i \in Li) M(c,i) \wedge c \leq g^-(n)\})$ for all $n \in I^+$.

We analyse the properties of the sets $MING^+(n)$ and $MING^-(n)$ in order to explain why they can be used in nearest-neighbour classification. We start with the set $MING^+(n)$. We have found the unique correspondence between instances n and the maximal boundary elements $g^+(n)$. Thus, since the concept languages used are UPL it is possible to prove that:

$$(\forall n_1, n_2 \in Li)((n_1 \neq n_2) \leftrightarrow (MING^+(n_1) \neq MING^+(n_2))).$$

This equivalence represents a nice property of the sets $MING^+(n)$, namely that the sets are unique for the instances, and the instances are unique for the sets. In addition to that there exists a second property that follows from the definition of the sets $MING^+(n)$. The property states that the elements of each set $MING^+(n)$ taken together cover less instances than the corresponding maximal element $g^+(n)$.

By analogy the same properties can be derived for the sets $MING^-(n)$. Thus, we conclude that the sets $MING^+(n)$ and $MING^-(n)$ represent training instances and can be used in nearest-neighbour classification.

Given the distance function D, the instance-based scheme G is an adaptation of the k-nearest-neighbour algorithm. When an instance $i \in Li$ has to be classified its k nearest sets $MING^+(n)$ and $MING^-(n)$ are found according to the distance function $D(S,i)$. Then the concept which sets form the majority among the k nearest sets is determined. The found concept is the classification of the instance i.

The classification of every instance, determined by the scheme G, can be explained by a simple algorithm. The algorithm forms sets G_c^+ and G_c^- from those elements of the nearest sets $MING^+(n)$ and $MING^-(n)$, respectively, that do cover the instance. Since the concept languages are UPL the algorithm generates the least upper bounds of the sets G_c^+ and G_c^-. The upper bounds are used in order to facilitate the user to understand the resulting classification of any instance.

The inductive bias of the scheme G is a preference bias. It can be determined analogously to the inductive bias of the scheme S.

8.2.3 Generalising the Schemes S and G

The scheme S can be easily generalised for a set T of target concepts, where $|T| > 2$. In this case for each concept $tc \in T$ we have to learn IBBS:$\langle\{S^{tc}(\{p\},\emptyset)\}_{p\in I^{tc}},\emptyset\rangle$ of the positive version space $VS^{tc}(\emptyset)$. Hence, applying the adapted k-nearest neighbour algorithm with the distance function $D(S,i)$ remains the same.

The scheme G can be easily generalised for a set T of target concepts as well, where $|T| > 2$. In this case for each concept $tc \in T$ we have to learn IBBS:$\langle\{S^{tc}(p,\{n\})\}_{p\in I^{tc}},\{G^{tc}(n)\}\rangle$ of version spaces $VS^{tc}(n)$ where $n \in \bigcup_{t\in T-\{tc\}} I^t$. Hence, applying the adapted k-nearest-neighbour algorithm with the distance function $D(S,i)$ remains the same.

In spite of that the use of the generalised instance-based scheme G when $|T| > 2$ is not desirable. To see why, consider an instance $n \in Li$ that is a positive instance of a concept $tc \in T$ and is negative for all other concept $t \in T - \{tc\}$. Thus, for each concept $t \in T - \{tc\}$ the maximal boundary set $G^t(n)$ has to be formed. If all the maximal boundary sets $G^t(n)$ are nonempty then by corollary 6.14 they are equal (since the concept languages used are UPL). This implies that all the sets $MING^t(n)$ are equal as well. If we generalise this result for $|T| > 2$, then concepts $t \in T$ have common sets $MING^t(n)$. This substantially decreases the classification performance of the scheme G. The decrease is due to a well-known property of the

k-nearest-neighbour algorithm: it cannot accurately classify new instances when the training sets contain the same instances with different training classifications.

The generalised instance-based classification scheme S does not have the mentioned drawback of the generalised scheme G in the general case. This is due to the fact that every instance p of an arbitrary concept $t \in T$ causes the corresponding minimal boundary set $S^t(\{p\}, \emptyset)$ to be added to the instance-based boundary sets of the version space $VS^t(\emptyset)$ of the concept t only. If we generalise this result for $|T| > 2$, then the concepts $t \in T$ do not have common sets $S^t(\{p\}, \emptyset)$. Thus, the classification performance of the generalised classification scheme S does not decrease when the number $|T|$ of target concepts increases[2].

8.2.4 Condensed Nearest-Neighbour Algorithm

The drawbacks (1d) and (2d) of the k-nearest-neighbour algorithm from section 8.1 certainly are problems for instance-based classification schemes S and G. Consider tables 8.1 and 8.2. In the context of positive version spaces for each positive instance p we keep a minimal boundary set $S(\{p\}, \emptyset)$. In the context of version spaces with respect to negative instances for each negative instance n we keep at least the maximal element $g(n)$ and we compute the set $MING(n)$. That is why instance-based classification with conjunctive version spaces is characterised with large memory requirements, and thus the time for classification is high.

To avoid these problems we propose to apply the condensed nearest-neighbour algorithm (CNN) (Hart, 1968). Given an instance-based learning task (see figure 8.1), the algorithm finds for each target concept $t \in T$ a subset $I^{t\prime}$ of the training set I^t such that every instance in $\bigcup_{t\in T} I^t$ is closer to an instance in $\bigcup_{t\in T} I^{t\prime}$ of the same concept than to an instance in $\bigcup_{t\in T} I^{t\prime}$ of a different concept. In this way, the instances in $\bigcup_{t\in T} I^{t\prime}$ can be used to classify all the instances in the training sets $\bigcup_{t\in T} I^t$ correctly.

The algorithm begins by initialising the subsets $I^{t\prime}$ of the target concepts $t \in T$. Each subset $I^{t\prime}$ is set equal to randomly-selected instance from the corresponding set I^t. Then every instance in $\bigcup_{t\in T} I^t$ is classified using only the instances in $\bigcup_{t\in T} I^{t\prime}$. If an instance $i \in I^t$ is misclassified, it is added to the subset $I^{t\prime}$. This guarantees that the instance will be classified correctly. The process of forming the subsets $I^{t\prime}$ stops when there are no instances in $\bigcup_{t\in T} I^t$ that are misclassified. Thus, the algorithm ensures that all instances in $\bigcup_{t\in T} I^t$ are classified correctly, though it does not guarantee the minimality of the subsets $I^{t\prime}$.

The CNN algorithm can be easily adapted to the schemes S and G for

[2] Note that the scheme S can face partially the problems of the generalised scheme G for a particular case. Consider a training instance $p_1 \in Li$ that belongs to a concept t_1 and a training instance $p_2 \in Li$ that belongs to a concept t_2. It can happen that $S^{t1}(\{p_1\}, \emptyset) = S^{t2}(\{p_2\}, \emptyset)$. Hence, some concepts can share common sets $S(\{p\}, \emptyset)$. The level of sharing is lower than that of the scheme G. Thus, even in this worst case the classification performance of the scheme S is still better than that of the generalised scheme G.

instance-based classification with conjunctive version spaces: the sets $S(\{p\}, \emptyset)$ and $MING(n)$ are treated as instances, and the distance function D is changed to a new one D_d. Given a UPL concept language Lc and two sets $S_1, S_2 \subseteq Lc$, the function D_d measures the number of descriptions in S_1 that are not subsumed by any description from S_2 in the partially-ordered structure of the concept language used. More formally:

$$D_d(S_1, S_2) = |\{s_1 \in S_1 | \neg(\exists s_2 \in S_2)(s_1 \leq s_2)\}|.$$

The CNN algorithm and its adaptation to the IBBS are especially sensitive to noise. This is due to the fact that noisy instances are usually misclassified by their neighbours, and thus they are retained.

8.2.5 Edited Nearest-Neighbour Algorithm

The drawback (3d) (see section 8.1); i.e., the noise sensitivity, of the k-nearest-neighbour algorithm is one of the main practical problems of the instance-based classification schemes S and G. If some of the training instances are noisy, then their corresponding sets $S(\{p\}, \emptyset)$, and $MING(n)$ are not correct. Hence, the classification performance of both schemes S and G degrades.

To avoid this problem we propose to employ the edited nearest-neighbour algorithm (ENN) given by Wilson (1972). Given an instance-based learning task (see figure 8.1), the algorithm is quite straightforward: it removes instances from the training sets that do not agree with the majority of their k nearest neighbours (where k is typically equal to 3). This edits out noisy instances as well as close border cases, leaving smoother decision boundaries.

The ENN algorithm can be easily adapted to the schemes S and G for instance-based classification with conjunctive version spaces using the same adaptation system from the previous subsection.

8.2.6 System based on Classification Scheme S

We have implemented in Java2 a system SS that is based on the generalised scheme S of instance-based classification with conjunctive version spaces. It has been implemented for the class of 1-DNF languages with discrete attributes (i.e., the conjunctive version spaces are defined in conjunctive extensions of these languages). The system has been programmed to learn from and to classify instances when we have at least two target concepts.

The system SS maintains the IBBS of the positive version spaces $VS^t(\emptyset)$ for each target concept t. It does not maintain any representation of version spaces with respect to negative instances for each target concept t. Thus, the system SS

does not support any basic conjunctive version-space operation. In contrast, the system does support the following three simple operations[3]:

- *Update*$_{SS}$: if a new training instance $\mathfrak{p}$ of a target concept t is given, the operation adds a new minimal boundary set $S^t(\{\mathfrak{p}\},\emptyset)$ to the IBBS:$\langle\{S^t(\{p\},\emptyset)\}_{p\in I^t},\emptyset\rangle$ of the positive version space $VS^t(\emptyset)$ of that concept t;

- *Retraction*$_{SS}$: if an instance $\mathfrak{p}$ of a target concept t is retracted, the operation removes the minimal boundary set $S^t(\{\mathfrak{p}\},\emptyset)$ from the IBBS:$\langle\{S^t(\{p\},\emptyset)\}_{p\in I^t},\emptyset\rangle$ of the positive version space $VS^t(\emptyset)$ of that concept t;

- *Classify*$_{SS}$: if an instance i has to be classified, the operation uses the scheme S to classify the instance. The operation has a parameter k that is the parameter of the nearest-neighbour algorithm.

The operations *Update*$_{SS}$ and *Retraction*$_{SS}$ are simplified versions of the conjunctive version-space operations *Update*$_{CVS}$ and *Retraction*$_{CVS}$. Thus, from section 7.2 it follows that the IBBS are heuristically adequate for these two operations when they are heuristically adequate for the basic version-space operations.

To determine the heuristical adequacy of the IBBS for the operation *Classify*$_{SS}$ we derive the worst-case time complexity of the algorithm of the operation. The complexity is derived in terms of the complexity analysis of the IBBS (see chapter 6) under the assumption that we have only two target concepts (+/−). When an instance i has to be classified then the operation *Classify*$_{SS}$ checks P boundary sets $S^+(\{p\},\emptyset)$ and N boundary sets $S^-(\{p\},\emptyset)$ whether they cover the instance. To determine whether a set $S^+(\{p\},\emptyset)$ or a set $S^-(\{p\},\emptyset)$ covers the instance we need Σ_p comparisons between elements of the sets and the instance. Thus, the overall worst-case time complexity is $O((P+N)\Sigma_p t_c)$. From the results of chapter 6 we conclude that the IBBS are heuristically adequate for the operation *Classify*$_{SS}$ when they are heuristically adequate for the basic version-space operations.

8.2.7 Experiments

The classification performance (CP) of the system *SS* has been tested on 8 datasets from the Machine Learning Database Repository at the University of California, Irvine (Blake and Merz, 1998). The parameter k of the operation *Classify*$_{SS}$ has been set equal to one. Ten-fold cross-validation has been used for all experiments. The results are shown in table 8.3.

[3] To distinguish the operations from that of conjunctive version spaces we label them with subscript *SS*.

Dataset	C	A	I	Majority	CP
cars	4	6	1728	70.0%	92.8 ± 2.3%
hepatitis	2	20	155	79.3%	80.6 ± 1.7%
iris	3	4	150	33.3%	91.8 ± 0.8%
monks-1	2	6	124	50.0%	78.3 ± 1.7%
monks-2	2	6	168	61.5%	56.6 ± 2.2%
monks-3	2	6	122	50.0%	86.3 ± 1.2%
tic-tac-toe	2	9	958	65.4%	98.5 ± 0.4%
vote	2	16	435	61.4%	92.9 ± 0.3%

Table 8.3: Classification Performance (CP) of the System *SS* (C is the number of target concepts; A is the number of attributes; I is the size of training data; Majority is the percentage of instances in the majority class).

In order to find the place of the system *SS* in terms of classification performance we compare it to a well-known environment in machine learning: the C4.5 decision-tree learner (Quinlan, 1993) (see table 8.4). The comparison shows that our system outperforms C4.5 in 5 out of 8 tasks. The classification performance of our system is 84.7 % while that of C4.5 is 82.9 %. The results are not a surprise since our system is essentially a version of the k-nearest-neighbour algorithm that has comparable performance with that of C4.5 (Domingos and Pazzani, 1996).

Dataset	CP
cars	92.3 ± 2.3 %
hepatitis	78.7 ± 4.7%
iris	93.4 ± 2.4%
monks-1	76.3 ± 1.5%
monks-2	53.2 ± 2.1%
monks-3	87.4 ± 2.2%
tic-tac-toe	85.6 ± 1.1%
vote	96.3 ± 1.3%

Table 8.4: Classification Performance (CP) of the C4.5 Decision-Tree Learner.

We have enriched the system *SS* with two additional components. The first one is the condensed nearest-neighbour algorithm (CNN). In table 8.5 we compare the classification performance of the system *SS* with that of the combination of the system *SS* and the CNN algorithm. The classification performance has been measured using ten-fold cross-validation when the parameter k of the operation Classify_{SS} has been set to one. The comparison shows that when the system *SS* is combined with the CNN algorithm its average predictive accuracy drops from 84.7% to 80.6%, but it requires in average only 24.68% of the size of the original IBBS.

Dataset	CP (*SS*)	%	CP (*SS*+CNN)	%
cars	92.8 ± 2.3%	100%	89.0 ± 1.0%	17.4%
hepatitis	80.6 ± 1.7%	100%	72.7 ± 2.0%	32.2%
iris	91.8 ± 0.8%	100%	87.6 ± 1.5%	20.2%
monks-1	78.3 ± 1.7%	100%	78.6 ± 1.0%	31.5 %
monks-2	56.6 ± 2.2%	100%	56.2 ± 3.1	44.8 %
monks-3	86.3 ± 1.2%	100%	76.9 ± 2.0%	27.3 %
tic-tac-toe	98.5 ± 0.4%	100%	93.2 ± 1.2%	14.2 %
vote	92.9 ± 0.3%	100%	90.9 ± 0.6%	9.6%

Table 8.5: Classification Performance (CP) of the System *SS* and the Combination of the System *SS* with the CNN Algorithm. The left column shows classification performance and the right column (%) shows what percent of the size of the IBBS that has been retained.

The second component used to enrich the system *SS* has been the edited nearest-neighbour algorithm (ENN). In order to compare the behaviour of the system *SS* and the combination of the system *SS* with the ENN algorithm the experiments have been repeated with 10% uniform class noise artificially added to each dataset. This was done by randomly changing the training classification of 10% of the instances in the training set to an incorrect value. The classification of the instances in the test set is not noisy, so the results indicate how well the system is able to predict the correct output even if some of the training instances are mislabeled.

Dataset	CP (*SS*)	%	CP (*SS*+ENN)	%
cars	86.9 ± 1.2%	100%	87.0 ± 1.7%	77.2%
hepatitis	67.5 ± 1.4%	100%	78.7 ± 1.4%	68.2%
iris	78.4 ± 1.1%	100%	81.8 ± 1.3%	73.9%
monks-1	72.2 ± 2.0%	100%	70.3 ± 3.0%	70.3%
monks-2	56.4 ± 2.3%	100%	59.3 ± 2.3%	56.2%
monks-3	78.3 ± 1.7%	100%	82.9 ± 1.0%	78.4%
tic-tac-toe	89.0 ± 0.7%	100%	93.2 ± 1.5%	84.4%
vote	88.3 ± 0.7%	100%	91.4 ± 0.6%	82.3%

Table 8.6: Classification Performance (CP) of the System *SS* and the Combination of the System *SS* with the ENN Algorithm. The left column shows classification performance and the right column (%) shows what percent of the size of the IBBS that has been retained.

Table 8.6 shows the classification performance and average storage requirements of the system *SS* and the combination of the system *SS* with the ENN algorithm over the same 8 datasets with 10% noise. The classification performance has been measured using ten-fold cross-validation when the parameter k of the operation

Classify$_{SS}$ has been set to one and the parameter k of the ENN algorithm has been set to three. The results are the following: the average classification performance of the system SS is 77.1%. The average classification performance of the system SS combined with the ENN algorithm is 80.5% and the average size of the IBBS retained is 73.8%.

To complete the experiments we have measured the running time of the algorithms of the operations *Update*$_{SS}$, *Retraction*$_{SS}$ and *Classify*$_{SS}$ on a computer with a Pentium II-333MHz processor and 128MB RAM.

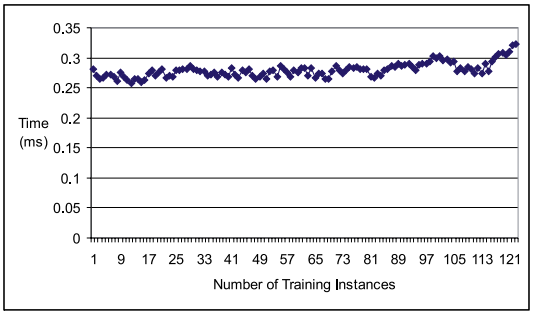

Figure 8.2: The Average Running Time of the Algorithm of the Operation *Update*$_{SS}$.

The average running time of the algorithm of the operation *Update*$_{SS}$ grows between 0.25 milliseconds and 0.35 milliseconds (see figure 8.2). We have expected the time to be constant since for each new training instance $\mathfrak{p}$ we only add a set $S^t(\mathfrak{p}, \emptyset)$ to the IBBS. The reason for the linear behaviour of the running time seems to be in the implementation of the system SS (e.g. the class Vector in Java2).

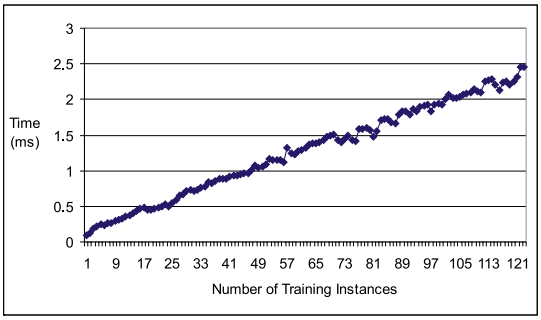

Figure 8.3: The Average Running Time of the Algorithm of the Operation *Retraction*$_{SS}$.

The average running time of the algorithm of the operation *Retraction*$_{SS}$ grows linearly from 0.1 milliseconds to 2.5 milliseconds (see figure 8.3). The linearly grow-

ing behaviour of the running time is due to the fact that the set $S^t(\mathfrak{p}, \emptyset)$ of the instance $\mathfrak{p}$ to be retracted has to be found.

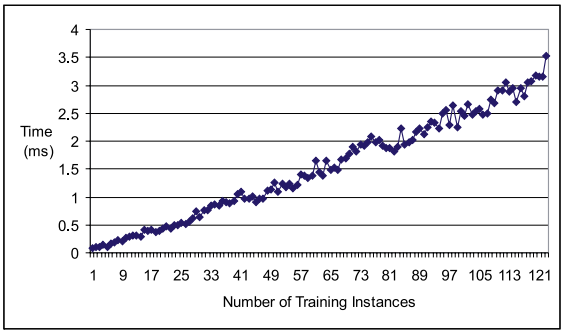

Figure 8.4: The Average Running Time of the Algorithm of the Operation $Classify_{SS}$.

The average running time of the algorithm of the operation $Classify_{SS}$ grows linearly between 0.1 milliseconds and 3.5 milliseconds (see figure 8.4). The linearly growing behaviour of the running time is due to the fact that each set $S^t(\mathfrak{p}, \emptyset)$ of the IBBS for all concepts $t \in T$ has to be visited.

8.2.8 System based on Classification Scheme G

We have implemented in Java2 a system SG that is based on the generalised scheme G of instance-based classification with conjunctive version spaces. It has been implemented for the class of 1-DNF languages with discrete attributes. The system has been programmed to learn from and to classify instances when we have at least two target concepts.

By analogy with the system SS, the system SG has the same set of operations that function with the IBBS of version spaces $VS^t(n)$ with respect to negative instances. It is easy to prove that the IBBS are heuristically adequate for these operations when they are heuristically adequate for the basic version-space operations and the operation that computes the sets $MING(n)$.

The system SG has passed the same experiments as the system SS. The resulting classification performance of the system is similar to that of the system SS for binary concept-learning tasks (datasets) (see tables 8.3, 8.5 and 8.6). When the system SG has been tested on datasets *iris* and *cars*, that have 3 and 4 target concepts, respectively, its classification performance has dropped 5 to 12 percent in comparison with the system SS. This has experimentally confirmed our expectations discussed in subsection 8.2.2: the use of the generalised scheme G has to be avoided when the number of target concepts is greater than two.

8.2.9 Discussion

The instance-based classification schemes S and G do not require any change in the IBBS representations of positive version spaces and version spaces with respect to negative instances. (Only the scheme G presupposes extra computation.) This means that the schemes do not require any change in the IBBS representations of conjunctive version spaces. Therefore, the classification schemes S and G can be added to the conjunctive version-space abstract data type when it is implemented using instance-based boundary sets. In this way when the conjunctive extensions of UPL concept languages are not complete; i.e., we have a restriction bias, the algorithm of the basic conjunctive version-space operation $Classify_{CVS}$ is used (see chapter 4). Otherwise, we can employ the instance-based classification schemes S and G that have preference biases.

The experimental analysis of the classification schemes S and G shows that they have a good classification performance on the used datasets. This is due to the fact that both schemes are based on the k-nearest-neighbour algorithm. Moreover, both schemes allow the use of additional algorithms with explanatory capabilities.

An interesting issue is when to apply the generalised scheme S and when to apply the generalised scheme G. If we have a binary concept-learning task the decision has to be based on the relevance of the elements of the sets $S^t(p, \emptyset)$ and $MING^t(n)$ for the task since the k-nearest-neighbour algorithm is sensitive with respect to irrelevant attributes. In all other cases the generalised scheme S is preferred since its classification performance does not degrade with an increasing number of target concepts.

The experimental results show that the combinations of the schemes S and G, and the CNN algorithm are capable to decrease significantly the size of the IBBS of conjunctive version spaces but with a price of a significant drop of the classification performance. This contrasts with the combinations of the schemes with the ENN algorithm. It has been experimentally proven that they are robust with respect to noise in the training instances and additionally reduce the size of the IBBS of conjunctive version spaces. Thus, we can conclude that the fourth part of our problem statement, namely the problem with noisy training instances is solved.

8.3 Instance-Based Classification with Disjunctive Version Spaces

Instance-based classification with disjunctive version spaces, represented in intersection preserving languages, can be obtained by duality from the section 8.2. In order to avoid repetitions we leave out its description.

8.4 Comparison with Related Work

In this section we compare the instance-based classification schemes S and G with relevant work on version spaces that aimed at overcoming the problem with incomplete concept languages and the problem with noisy training instances.

8.4.1 Extended Version Spaces

Mitchell (1978) proposed an extended version-space approach for learning in "less perfect situations". The "less perfect situations" are situations in which there is noise in the training instances and the descriptions of target concepts are not presented in the concept languages used.

The extended version-space approach is based on a special type of version spaces $VS_{s,g}$ defined as sets of concept descriptions consistent with $|I^+| - s$ positive instances and $|I^-| - g$ negative instances. More formally:

$$VS_{s,g} = \{c \in Lc | (\exists I^{+\prime} \in SS(I^+, |I^+| - s))(\exists I^{-\prime} \in SS(I^-, |I^-| - g)) cons(c, \langle I^{+\prime}, I^{-\prime} \rangle)\}$$

where $SS(A, m) = \{X \subseteq A | |X| = m\}$.

The approach maintains all version spaces $VS_{s,g}$ which parameters s and g belong to the preliminary given intervals [0,sn] and [0,gn]. The instance classification is realised using those version spaces $VS_{s,g}$ that are nonempty and consistent with maximal number of the training instances. This means that the instance classification does not take into account s positive and g negative training instances. Hence, if some of these instances are noisy, the approach can handle the noise in the training data.

The approach can be used for learning target concepts that are not represented in the concept languages used. The key idea is to minimise the parameter s when the parameter gn is set to zero. In this case the version spaces $VS_{s,g}$ contain concept descriptions that are consistent with an approximately maximal number of positive training instances and all the negative training instances. Hence, the descriptions in the version spaces $VS_{s,g}$, considered in disjunction, can represent the target concepts. By duality, an analogous solution exists when the parameter g is minimised with the parameter sn set to zero. In this case, the descriptions in the version spaces $VS_{s,g}$, considered in conjunction, are assumed to represent the target concepts[4].

The first problem of the extended version-space approach is computational: instead of updating one version space a set of version spaces, represented by boundary sets, is updated. Thus, the computational problem of boundary sets is multiplied. The second problem is how to set the parameters sn and gn since the target concepts are not known in advance as well as the level of noise in the training instances. In

[4] Note that the latter solution has not been given in (Mitchell, 1978).

spite of that the approach has been used in the learning component of the Meta-Dendral program (Mitchell, 1978).

The extended version-space approach can be compared with the instance-based classification schemes S and G of conjunctive and disjunctive version spaces. The comparison shows that the classification schemes have lower complexities since the size of the IBBS do not grow exponentially in the number of the training instances in general. In addition, both schemes require only one parameter k (of the nearest-neighbour algorithm) that can be adjusted using standard techniques from instance-based learning (Wilson and Martinez, 2000).

8.4.2 Incremental Version-Space Merging

Hirsh (1989) proposed to consider version-space learning as a process of incremental version-space merging (IVSM). The scenario is simple: the initial version space of a target concept is set equal to the concept language used. When a new instance is given its version space is formed from those concept descriptions that are consistent with the instance. The instance's version space is intersected with the version space of the target concept. The resulting version space is consistent with the previous training instances and the new instance, and thus it is the version space of the target concept so far. The learning process continues till no instances are available or the version space collapses.

The IVSM approach was extended to handle training data with bounded inconsistency. "The underlying assumption for this class of inconsistency is that some small perturbation to the description of any bad instance will result in a good instance" (Hirsh, 1989, p.25). That is why when a new training instance is given it is perturbed so that all its possible interpretations in the context of the current concept-learning task are found. The instance's interpretations are used for formation of the version space of the instance. It is defined as a set of concept descriptions consistent with at least one instance's interpretation. The version space of the target concept is intersected with the version space of the instance as usually. If in some point of the concept-learning process the version space is not empty then it contains descriptions consistent with at least one chain of interpretations of all the training instances. Therefore, the version spaces in the extended IVSM approach can handle data with bounded inconsistency.

A complexity analysis shows that the extended IVSM approach can be practical since it overcomes some of the computational problems of the extended version-space approach, but not all. For example, the version space of a target concept can be a union of the version spaces that are consistent with different chains of the interpretations of the training instances. This increases the sizes of the boundary sets of the version space; i.e., the computational problem of the boundary sets can be much more difficult. In addition to that, we have to mention that in order to use the extended IVSM approach in practice we have to know how to perturb instances. This is not a trivial task that varies from domain to domain.

All these conclusions weaken the extended IVSM approach, when it is compared with the instance-based classification schemes S and G of conjunctive and disjunctive version spaces. The comparison shows that both schemes have lower computational complexity in general since the IBBS do not grow exponentially in the number of the training instances. Moreover, the schemes do not require perturbing the training instances that simplifies their application.

8.4.3 Murray's Disjunctive Version Spaces

Murray (1987) proposed a disjunctive version space approach for concept-learning tasks characterised by insufficient completeness of concept languages. Murray's disjunctive version spaces were defined as sets of maximally complete version spaces. A version space is maximally complete if it is a set of concept descriptions that are consistent with a maximal number of the positive training instances and all the negative training instances. The latter definition and the structure of Murray's disjunctive version spaces determine the classification procedure: an instance is classified as positive if at least one maximally complete version space classifies it as positive; an instance is classified as negative if all the maximally complete version spaces classify it as negative; otherwise the instance classification cannot be determined. To determine whether a version space classifies an instance the rule of the unanimous vote of the descriptions in version spaces is used.

If Murray's disjunctive version spaces are considered in the disjunctive extensions of concept languages, it is possible to prove that they are subsets of the disjunctive version spaces defined in chapter 4. Thus, the inductive bias of Murray's disjunctive version spaces is a combination of restriction and preference biases.

Since Murray considered the case of attributive languages only, Sablon (1995) and Smirnov and Neves (1998) generalised his disjunctive version spaces for the whole class of admissible concept languages. Sablon extended the disjunctive version spaces for the description identification problem (Mellish, 1991) while Smirnov and Neves extended the disjunctive version spaces in the context of concept drift (Kubat, 1991).

An analysis of Murray's disjunctive version spaces shows their two principal drawbacks. The first one is computational. Instead of one version space a set of maximally complete version spaces is maintained. Since each maximally complete version space is represented by boundary sets, the computational problem of the boundary sets can be multiplied. The second drawback of Murray's disjunctive version spaces is that noisy training instances remove descriptions of the target disjunctions. Thus, Murray's disjunctive version spaces are sensitive with respect to noisy training instances.

The research presented in the thesis has actually started from the drawbacks of Murray's disjunctive version spaces. To avoid the first drawback we have proposed the IBBS to represent conjunctive and disjunctive version spaces. This allows us to apply the instance-based classification schemes S and G introduced in this chapter. The extensions of the schemes with the edited nearest-neighbour algorithm

are robust with respect to noisy training data. Thus, the second drawback can be overcome as well.

8.4.4 Sebag's Disjunctive Version Spaces

Sebag (1996) extended disjunctive version spaces for concept learning tasks characterised with noisy training instances. Sebag's disjunctive version spaces were defined as sets of version spaces with respect to positive training instances. The latter were proposed to be represented by a special layered representation that was proven to be tractable for conjunctive attributive languages. The layered representation coincides with the IBBS of a version space with respect to a positive instance.

The classification procedure was generalised for an arbitrary number of target concepts. An instance is classified to belong to a concept c if among all version spaces which layered representations cover the instance, the version spaces for positive instances of the concept c are the majority. Therefore, it is possible to show that in this case we have a preference bias.

In order to handle noise in training data Sebag proposed a procedure for classification with one version space with respect to a positive instance. The procedure has two parameters. The first one determines how many descriptions in one layer of maximal boundary sets have to cover an instance in order to classify this instance as positive by the layer. The second parameter determines how many layers have to cover an instance in order to classify this instance as positive by the version space.

The parameters were proposed to be adjusted globally for all version spaces with respect to positive instances by a process of trial and error. The training set is divided into two training sets. The first one (70%) is used for inducing the disjunctive version spaces. The second training set (30%) is used for adjusting the parameters.

The reported predictive classification performance on datasets from the Machine Learning Database Repository at the University of California is good.

The promising results on Sebag's disjunctive version spaces require a broader comparison with our research[5]. We start from a conceptual level: Sebag's disjunctive version spaces are different from our disjunctive version spaces since the latter require an additional negative version space for their representation. Therefore, our disjunctive version spaces can be used in classification based on the rule of the unanimous vote of descriptions in version spaces while Sebag's cannot. The price is obvious: our disjunctive version spaces require more space. But the price is good: we propose two instance-based classification schemes S and G that are almost parameter free, while Sebag's classification procedure of disjunctive version spaces requires two parameters that have to be adjusted.

[5] The comparison is made in terms of disjunctive version spaces. Note that by duality the disjunctive version spaces considered in this thesis are represented by the negative version space and version spaces with respect to positive instances.

8.4.5 Separate-and-Conquer Algorithms

Smirnov and van den Herik (2000) proposed to use the IBBS representations of positive and negative version spaces of conjunctive and disjunctive spaces, respectively, for designing separate-and-conquer algorithms (Fürnkranz, 1999). They considered two design schemes. For simplicity we give them in the context of conjunctive version spaces.

The first scheme was proposed for designing separate-and-conquer algorithms that learn in CNF extensions of the UPL concept languages. The algorithms start with an empty conjunction C. Then they "greedily" form conjuncts c of C. Each conjunct c is formed from these elements of the minimal boundary sets $S(p,\emptyset)$ of the positive version space $VS(\emptyset)$ that maximise a user supplied function *Perf* in a disjunction. Hence, each conjunct c is a disjunction of elements of the sets $S(p,\emptyset)$ chosen in this way. The process of generating conjuncts c stops when a given *stopping criterion* fires.

The second scheme was proposed for designing separate-and-conquer algorithms that learn in DNF extensions of the UPL concept languages. They start with a disjunction D with $|I^+|$ disjuncts. Each disjunct $C(p)$ of D is initialised equal to the conjunction of all elements of the corresponding set $S(p,\emptyset)$. The conjuncts s of $C(p)$ are "greedily" removed so that the disjunct $C(p)$ maximises a user supplied function *Perf*. Revising disjuncts $C(p)$ stops when a given *stopping criterion* fires.

In principle both design schemes can be considered as a practical methodology for constructing separate-and-conquer algorithms based on the IBBS. Note that separate-and-conquer algorithms implement preference biases. Therefore, the design schemes are methods for applying preference biases on the IBBS of conjunctive and disjunctive version spaces.

The separate-and-conquer algorithms, constructed according to the design schemes, have two principal drawbacks. The first one is that the IBBS structures of the rules, induced by separate-and-conquer algorithms, are different from the IBBS themselves. This implies that no basic conjunctive or disjunctive version-space operations can be realised with the rule structures. The second drawback is that when training instances are retracted the separate-and-conquer algorithms require substantial re-processing of the used data.

The drawbacks weaken the considered separate-and-conquer algorithms when they are compared with the instance-based classification schemes S and G. The latter do not require forming additional classification structures and do not change the IBBS representations of conjunctive and disjunctive version spaces.

8.5 Chapter Conclusion

This chapter has considered the problem of noisy training instances. It has proposed to apply the k-nearest-neighbour algorithm on the instance-based boundary sets of

the conjunctive and disjunctive version spaces. This has resulted in two instance-based classification schemes S and G based on preference biases. Both schemes have been combined with the edited nearest-neighbour algorithm. It has been shown that the combinations are robust with respect to noise in training data. Thus, *the main contribution of this chapter is that the fourth part of our problem statement, namely the problem with noisy training instances, has been solved for the conjunctive and disjunctive version spaced represented by instance-based boundary sets.*

Chapter 9

Conclusions

This final chapter summarises the results and chapter conclusions of the thesis. For a proper understanding we reiterate the problem statement from chapter 1.

Do there exist version spaces and their corresponding representations that can simultaneously solve:

(1) the problem with incomplete concept languages;

(2) the computational problem;

(3) the retraction problem; and

(4) the problem with noisy training instances?

In sections 9.1 to 9.3 all the four parts of the problem statement are re-addressed and evaluated[1]. The results are summarised in section 9.4. Finally, section 9.5 provides promising directions for future research.

9.1 Conjunctive and Disjunctive Version Spaces

In chapter 4 we have dealt with the problem of incomplete concept languages. The key idea has been to extend concept languages using either logical conjunction or disjunction. Conjunctive and disjunctive version spaces have been defined as version spaces in conjunctive and disjunctive extensions of concept languages. Their completeness properties have been analysed. We have proven that conjunctive and disjunctive version spaces are more complete than (ordinary) version spaces. This

[1] This is realised using the results of chapters 4, 6, 7 and 8. Note that chapters 2, 3, and 5 have an introductory character.

implies that they are solutions of more concept-learning tasks. Therefore, we have concluded that:

Conjunctive and disjunctive version spaces are a solution to the problem of incomplete concept languages, the first part of our problem statement, when conjunctive or disjunctive extensions of concept languages are sufficiently complete with respect to the concept-learning tasks to be solved.

In the second part of chapter 4 we have proposed to consider conjunctive and disjunctive version spaces as abstract data types. It has been proven that conjunctive and disjunctive version spaces can be represented by ordinary version spaces as well as that the basic conjunctive and disjunctive version-space operations can be expressed in terms of the basic version-space operations. Therefore, we have concluded that *the conjunctive and disjunctive version-space abstract data types can be implemented via the version-space abstract data type.*

9.2 Instance-Based Boundary Sets

In chapter 6 we have proposed a new version-space representation, called instance-based boundary sets, that can be used for implementing the version-space abstract data type. It has been proven that the representation is epistemologically adequate for the basic version-space operations *Initialise, Update, Retraction, Collapsed?, Classify, Member?, Intersection, Subset?* and *Equal?* for the classes of intersection-preserving and union-preserving concept languages. We have derived the conditions when the instance-based boundary sets are heuristically adequate for the operations and when they are tractable. Analysing the conditions we have established that they can be easily met in practice. Therefore, we have concluded that:

If the concept languages are intersection-preserving or union-preserving, the instance-based boundary sets can provide a solution to the computational problem and the retraction problem that correspond to the second and third part of our problem statement[2].

In chapter 7 we have combined the results of chapters 4 and 6. We have shown that the instance-based boundary sets can represent conjunctive and disjunctive version spaces. Moreover, we have demonstrated that this representation is epistemologically adequate with respect to the basic conjunctive and disjunctive version-space operations for the classes of intersection-preserving and union-preserving languages. Thus, we have shown that the conjunctive and disjunctive version-space abstract data types can be implemented using instance-based boundary sets. For this reason

[2]The only version space representation so far that is efficient for the operation *Retraction* is the representation of Idemstam-Almquist (1990) that is applicable for conjunctive languages with tree-structured attributes. Note that the conjunctive languages with tree-structured attributes are a sub-class of the intersection-preserving concept languages.

we have determined the conditions when this representation is heuristically adequate for the basic conjunctive and disjunctive version-space operations as well as the conditions when it tractably represents conjunctive and disjunctive version spaces. Therefore, we have concluded that:

If the concept languages are union-preserving or intersection preserving, the conjunctive and disjunctive version spaces represented by instance-based boundary sets can provide a solution to the problem with incomplete concept languages, the computational problem, and the retraction problem, that correspond to the first, second, and third part of our problem statement.

9.3 Instance-Based Classification

In chapter 8 we have presented two instance-based classification schemes, based on the instance-based boundary-set representation of conjunctive and disjunctive version spaces. It has been shown that the schemes are applicable for the classes of intersection-preserving and union-preserving concept languages. We have analysed their inductive biases and determined that they implement preference biases. Nevertheless, the schemes do not require any change of the instance-based boundary-set representation of conjunctive and disjunctive version spaces. Thus, they have been proposed to be added to the conjunctive and disjunctive version-space abstract data types. The schemes have been combined with the edited nearest-neighbour algorithm. The empirical study of the combinations has shown that their classification performance is robust with respect to noise in the training instances.

In addition to the instance-based classification schemes we have sketched in section 8.4 a second approach for applying preference biases with instance-based boundary sets. The approach is a methodology for designing a broad spectrum of separate-and-conquer algorithms that are based on instance-based boundary sets.

With this background we conclude that:

The instance-based boundary sets can be used for designing learning procedures that implement different types of inductive biases including those that can handle noise in training data. This forms our answer to the problem with noisy training instances that corresponds to the last, fourth part of our problem statement.

9.4 Summary of Conclusions

Taking into account the results shown in the previous sections we conclude that:

If the concept languages are intersection-preserving or union-preserving, the conjunctive and disjunctive version spaces represented by instance-based boundary sets can solve simultaneously:

(1) the problem with incomplete concept languages;

(2) the computational problem;

(3) the retraction problem; and

(4) the problem with noisy training instances.

9.5 Future Research

We envision three areas of promising research: (1) a further investigation of the conditions for the concept-language completeness, (2) a specification of preference biases for the instance-based boundary sets, and (3) a study of applicability of the instance-based boundary sets for the description identification task. We discuss them briefly below.

Completeness of Concept Languages. In chapter 4 we have proposed to extend concept languages with logical conjunction or disjunction in order to improve their completeness for the concept-learning tasks to be solved. The logical conjunction and disjunction do not exhaust the whole repertoire of logical operations. Therefore, it is worthwhile to investigate the applicability of different logical operations (separately or in combination) in order to improve the completeness of various classes of concept languages. The research, if successful, can result in new types of version spaces which' properties, representations, and operations require a new thorough investigation.

Preference Biases for Instance-Based Boundary Sets. In chapter 8 we have considered two approaches for applying preference biases with instance-based boundary sets when they represent conjunctive and disjunctive version spaces. A new research question can be: "Are there any other preference biases that can be applied on instance-based boundary sets?" If so, it is important to investigate the complexity of the procedures implementing these biases as well as whether they preserve the structure of the instance-based boundary sets.

Instance-Based Boundary Sets and the Description Identification Task. The description identification task is a concept-learning task that additionally to training instances has training concepts. Mellish (1991) showed that the complete solution of the task, specified by the consistency criterion, is a version space. He proposed to represent the version space with boundary sets and to apply a modification of the boundary-set update algorithm that is capable to learn from training instances and concepts. Sablon, DeRaedt, and Bruynooghe (1994) addressed the computational problem of the boundary sets in the context of the description identification task. They applied a depth first search procedure in order to decrease

the size of the boundary sets. Further, Sablon (1995) extended their framework to disjunctive version spaces. Since the number of all possible disjuncts can be exponential, the size of the boundary sets of the disjunctive version spaces can be exponential as well. Thus, the framework is not practical, it has only theoretical value.

To avoid the mentioned problems we started some research on the description identification with version spaces (Smirnov and Braspenning, 1999). We proposed to represent version spaces with instance-based boundary sets. The results are promising but we still have to specify the whole class of applicable concept languages. In addition to that, we plan to use the conjunctive and disjunctive version spaces as well as the classification procedures based on preference biases as proposed in the thesis. Looking forward into the future we may state that if the results of the three new research areas are successful we can build a complete applicable theory of version spaces.

Appendix A

Correctness of the Algorithms of the Basic Version-Space Operations based on Boundary Sets

This appendix consists of theorems for correctness of the algorithms of the basic version-space operations that are based on the boundary-set representation (Mitchell, 1978) (see chapter 5). The theorems are given for the class of admissible concept languages. Their proofs are shown when these proofs were not given in (Mitchell, 1978) or (Hirsh, 1989).

Theorems of the Algorithm of the Operation *Update*

Theorem A.1. (Revising the boundary sets given a positive training instance) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*; and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with associated version space* VS' *and boundary sets* $\langle S', G' \rangle$*, where* $I^{+\prime} = I^+ \cup \{\mathfrak{p}\}$*. Then the sets* S' *and* G' *obey the following equalities:*

$$S' = MIN(\{c \in Lc | (\exists s \in S)(\exists g \in G)((s \leq c) \wedge (c \leq g)) \wedge M(c, \mathfrak{p})\})$$
$$G' = \{g \in G | M(g, \mathfrak{p})\}.$$

Theorem A.2. (Revising the boundary sets given a negative training instance) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and*

boundary sets $\langle S, G\rangle$*; and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime}\rangle\rangle$ *with associated version space* VS' *and boundary sets* $\langle S', G'\rangle$*, where* $I^{-\prime} = I^- \cup \{\mathfrak{n}\}$*. Then the sets* S' *and* G' *obey the following equalities:*

$$G' = MAX(\{c \in Lc | (\exists s \in S)(\exists g \in G)((s \leq c) \wedge (c \leq g)) \wedge \neg M(c, \mathfrak{n})\})$$
$$S' = \{s \in S | \neg M(s, \mathfrak{n})\}.$$

Theorems A.1 and A.2 have two simple corollaries A.3 and A.4. The corollaries consider the case when we have a version space VS_1 based on training sets I_1^+ and I_1^-, and a second version space VS_2 based on training sets I_2^+ and I_2^-. The first corollary A.3 states that if $I_1^+ \subseteq I_2^+$ and $I_1^- = I_2^-$, then the maximal boundary set G_2 of VS_2 is equal to a subset of the maximal boundary set G_1 of VS_1 which elements cover all the positive instances of the subset $I_2^+ - I_1^+$. The second corollary A.4 states that if $I_1^+ = I_2^+$ and $I_1^- \subseteq I_2^-$, then the minimal boundary set S_2 of VS_2 is equal to a subset of the minimal boundary set S_1 of VS_1 which elements do not cover any negative instance of the subset $I_2^- - I_1^-$.

Corollary A.3. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^-\rangle\rangle$ *with associated version space* VS_1 *and boundary sets* $\langle S_1, G_1\rangle$*; and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^-\rangle\rangle$ *with associated version space* VS_2 *and boundary sets* $\langle S_2, G_2\rangle$*. If* $I_1^+ \subseteq I_2^+$ *and* $I_1^- = I_2^-$*, then the maximal boundary sets* G_1 *and* G_2 *obey the following equalities:*

$$G_2 = \{g \in G_1 | (\forall i \in I_2^+ - I_1^+) M(g, i)\}.$$

Proof. The proof follows from the proof of theorem A.1. □

Corollary A.4. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^-\rangle\rangle$ *with associated version space* VS_1 *and boundary sets* $\langle S_1, G_1\rangle$*; and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^-\rangle\rangle$ *with associated version space* VS_2 *and boundary sets* $\langle S_2, G_2\rangle$*. If* $I_1^+ = I_2^+$ *and* $I_1^- \subseteq I_2^-$*, then the minimal boundary sets* S_1 *and* S_2 *obey the following equalities:*

$$S_2 = \{s \in S_1 | (\forall i \in I_2^- - I_1^-) \neg M(s, i)\}.$$

Proof. The proof follows from the proof of theorem A.2. □

Theorems of the Algorithm of the Operation *Retraction*

Theorem A.5. (Revising the boundary sets given a positive training instance to be retracted) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*; and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with associated version space* VS' *and boundary sets* $\langle S', G' \rangle$*, where* $I^{+\prime} = I^+ - \{\mathfrak{p}\}$ *and* $\mathfrak{p} \in I^+$*. Then the sets* S' *and* G' *obey the following equalities:*

$$\begin{aligned} S' &= MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\})) \\ G' &= MAX(G \cup MAX(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\})). \end{aligned}$$

Proof. By theorem 3.7:

$$VS' = VS \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}.$$

According to definition 5.17 of the boundary sets it follows that:

$$\begin{aligned} S' &= MIN(VS') \\ &= MIN(VS \cup \{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\}) \\ &= MIN(MIN(VS) \cup MIN(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\})) \\ &= MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\})) \end{aligned}$$

Analogously, we prove that:

$$G' = MAX(G \cup MAX(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\})).$$

□

Theorem A.6. (Revising the boundary sets given a negative training instance to be retracted) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*; and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle\rangle$ *with associated version space* VS' *and boundary sets* $\langle S', G' \rangle$*, where* $I^{-\prime} = I^- - \{\mathfrak{n}\}$ *and* $\mathfrak{n} \in I^-$*. Then the sets* S' *and* G' *obey the following equalities:*

$$\begin{aligned} S' &= MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\})) \\ G' &= MAX(G \cup MAX(\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\})). \end{aligned}$$

Proof. The proof is dual to that of theorem A.5. It uses theorem 3.8. □

Theorems of the Algorithm of the Operation *Collapsed?*

Theorem A.7. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*. If the concept language Lc is admissible then:*

$$(VS \neq \emptyset) \leftrightarrow (S \neq \emptyset).$$

Proof. ($\rightarrow$) Since Lc is admissible, VS is bounded. Thus, according to definition 5.11 $VS \neq \emptyset$ implies $S \neq \emptyset$.

($\leftarrow$) According to definition 5.17 $S \subseteq VS$. Thus, $S \neq \emptyset$ implies $VS \neq \emptyset$.

□

Theorem A.8. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*. If the concept language Lc is admissible then:*

$$(VS \neq \emptyset) \leftrightarrow (G \neq \emptyset).$$

Proof. The proof is analogous to that of theorem A.7. □

Theorem of the Algorithm of the Operation *Converged?*

Theorem A.9. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*. Then:*

$$(|VS| = 1) \leftrightarrow ((S = G) \wedge (|S| = 1) \wedge (|G| = 1)).$$

Proof. ($\rightarrow$) According to definition 5.17 from $|VS| = 1$ we conclude that $VS = S = G$ and $|S| = 1 \wedge |G| = 1$.

($\leftarrow$) If $S = G$, then by theorem 5.18 since Lc is admissible $VS = S = G$. Thus, $(|S| = 1) \wedge (|G| = 1)$ imply $|VS| = 1$. □

Theorems of the Algorithm of the Operation *Classify*

Theorem A.10. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS and boundary sets* $\langle S, G \rangle$*. Then:*

$$(\forall i \in Li)((\forall c \in VS) M(c, i) \leftrightarrow (\forall s \in S) M(s, i)).$$

Proof. The statement of the theorem is equivalent to:

$$(\forall i \in Li)((\exists c \in VS)\neg M(c,i) \leftrightarrow (\exists s \in S)\neg M(s,i)).$$

Thus, we prove the theorem by proving the last equivalence.

($\rightarrow$) Consider an arbitrary $i \in Li$ such that there exists $c \in VS$ and $\neg M(c,i)$. By theorem 5.18 there exists $s \in S$ such that $s \leq c$. Since Lc is admissible, from $\neg M(c,i)$ and $s \leq c$ by lemma 5.6 we conclude that $\neg M(s,i)$. But i is arbitrarily chosen. Thus, we have proven that:

$$(\forall i \in Li)((\exists c \in VS)\neg M(c,i) \rightarrow (\exists s \in S)\neg M(s,i)),$$

and the first part of the theorem is proven.

($\leftarrow$) The second part of the proof follows from definition 5.17 stating that $S \subseteq VS$. □

Theorem A.11. *Consider a task $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ with associated version space VS and boundary sets $\langle S, G\rangle$. Then:*

$$(\forall c \in VS)\neg M(c,i) \leftrightarrow (\forall g \in G)\neg M(g,i).$$

Proof. The proof is dual to that one of theorem A.10. □

Theorem of the Algorithm of the Operation *Intersection*

Theorem A.12. *Consider a task $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ with associated version space VS_1 and boundary sets $\langle S_1, G_1\rangle$, and a second task $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ with associated version space VS_2 and boundary sets $\langle S_2, G_2\rangle$. Then the version space $VS_{12} = VS_1 \cap VS_2$ is represented by boundary sets $\langle S_{12}, G_{12}\rangle$ that obey the following equalities:*

$$S_{12} = \{c \in MinG(S_1, S_2) | (\exists g_1 \in G_1)(\exists g_2 \in G_2)((c \leq g_1) \wedge (c \leq g_2))\}$$
$$G_{12} = \{c \in MaxS(G_1, G_2) | (\exists s_1 \in S_1)(\exists s_2 \in S_2)((s_1 \leq c) \wedge (s_2 \leq c))\}$$

where

$$MinG(S_1, S_2) = MIN(\{c \in Lc | (\exists s_1 \in S_1)(\exists s_2 \in S_2)((s_1 \leq c) \wedge (s_2 \leq c))\})$$
$$MaxS(G_1, G_2) = MAX(\{c \in Lc | (\exists g_1 \in G_1)(\exists g_2 \in G_2)((c \leq g_1) \wedge (c \leq g_2))\}).$$

Theorem of the Algorithm of the Operation *Subset?*

Theorem A.13. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with associated version space* VS_1 *and boundary sets* $\langle S_1, G_1 \rangle$*, and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with associated version space* VS_2 *and boundary sets* $\langle S_2, G_2 \rangle$*. Then:*

$$(VS_1 \subseteq VS_2) \leftrightarrow ((\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1) \wedge (\forall g_1 \in G_1)(\exists g_2 \in G_2)(g_1 \leq g_2)).$$

Proof. ($\rightarrow$) Since Lc is admissible, the first part of the theorem follows from theorem 5.18.

($\leftarrow$) Consider an arbitrary $c \in VS_1$. Since VS_1 is represented by boundary sets S_1 and G_1, by theorem 5.18:

$$(\exists s_1 \in S_1)(\exists g_1 \in G_1)((s_1 \leq c) \wedge (c \leq g_1)).$$

Consider particular $s_1 \in S_1$ and $g_1 \in G_1$ such that $s_1 \leq c \leq g_1$. Since $(\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1)$ and $(\forall g_1 \in G_1)(\exists g_2 \in G_2)(g_1 \leq g_2)$, there exist $s_2 \in S_2$ and $g_2 \in G2$ such that $s_2 \leq s_1$ and $g_1 \leq g_2$. Thus,

$$s_2 \leq s_1 \leq c \leq g_1 \leq g_2.$$

But, the relation "$\leq$" is transitive. Hence, the last derivation implies:

$$s_2 \leq c \leq g_2.$$

Thus, by theorem 5.18:

$$c \in VS_2.$$

The second part of theorem is proven. □

Theorem of the Algorithm of the Operation *Equal?*

Theorem A.14. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with associated version space* VS_1 *and boundary sets* $\langle S_1, G_1 \rangle$*, and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2 *and boundary sets* $\langle S_2, G_2 \rangle$*. Then:*

$$VS_1 = VS_2 \leftrightarrow ((S_1 = S_2) \wedge (G_1 = G_2)).$$

Proof. The proof follows from theorem 5.18. □

Appendix B

Correctness of the Algorithms of the Basic Version-Space Operations based on Unilateral Boundary Sets

This appendix consists of theorems for correctness of the algorithms of the basic version-space operations that are based on the UBSS representation (Hirsh, 1992b) (see chapter 5). The theorems are given for the class of lower-admissible concept languages. The proofs of the theorems are not shown only in the case when they were given in (Hirsh, 1992b), or are analogous to the proofs from Appendix A.

Theorems of the Algorithm of the Operation *Update*

Theorem B.1. (Revising the UBSS given a positive training instance) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS represented by* $\langle S, I^- \rangle$*; and a second task* $\langle Li, Lc, M, \langle I^{+\prime}, I^- \rangle\rangle$ *with associated version space* VS' *represented by* $\langle S', I^- \rangle$*, where* $I^{+\prime} = I^+ \cup \{\mathfrak{p}\}$*. Then the set* S' *obeys the following equality:*

$$S' = MIN(\{c \in Lc | (\exists s \in S)(s \leq c) \wedge (\forall n \in I^-) M(c, n) \wedge M(c, \mathfrak{p})\}).$$

Theorem B.2. (Revising the UBSS given a negative training instance) *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS represented by* $\langle S, I^- \rangle$*; and a second task* $\langle Li, Lc, M, \langle I^+, I^{-\prime} \rangle\rangle$ *with associated version space*

VS' and boundary sets $\langle S', I^{-\prime}\rangle$, where $I^{-\prime} = I^- \cup \{\mathfrak{n}\}$. Then the set S' obeys the following equality:

$$S' = \{s \in S | \neg M(s, \mathfrak{n})\}.$$

Theorems of the Algorithm of the Operation *Retraction*

Theorem B.3. (Revising the UBSS given a positive training instance to be retracted) *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with associated version space VS represented by $\langle S, I^-\rangle$; and a second task $\langle Li, Lc, M, \langle I^{+\prime}, I^-\rangle\rangle$ with associated version space VS' represented by $\langle S', I^-\rangle$, where $I^{+\prime} = I^+ - \{\mathfrak{p}\}$ and $\mathfrak{p} \in I^+$. Then the set S' obeys the following equality:*

$$S' = MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ - \{\mathfrak{p}\}, I^- \cup \{\mathfrak{p}\}\rangle)\})).$$

Proof. The proof of the theorem coincides with the first part of the proof of theorem A.5. □

Theorem B.4. (Revising the UBSS given a negative training instance to be retracted) *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with associated version space VS represented by $\langle S, I^-\rangle$; and a second task $\langle Li, Lc, M, \langle I^+, I^{-\prime}\rangle\rangle$ with associated version space VS' represented by $\langle S', I^{-\prime}\rangle$, where $I^{-\prime} = I^- - \{\mathfrak{n}\}$ and $\mathfrak{n} \in I^-$. Then the set S' obeys the following equality:*

$$S' = MIN(S \cup MIN(\{c \in Lc | cons(c, \langle I^+ \cup \{\mathfrak{n}\}, I^- - \{\mathfrak{n}\}\rangle)\})).$$

Proof. The proof is dual to that of theorem B.3. □

Theorems of the Algorithm of the Operation *Collapsed?*

Theorem B.5. *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with associated version space VS represented by $\langle S, I^-\rangle$. Then:*

$$(VS \neq \emptyset) \leftrightarrow (S \neq \emptyset).$$

Proof. The proof coincides with that of theorem A.7. □

Theorems of the Algorithm of the Operation *Converged?*

Theorem B.6. *Consider a task $\langle Li, Lc, M, \langle I^+, I^-\rangle\rangle$ with associated version space VS represented by $\langle S, I^-\rangle$. Then:*

$$(|VS| = 1) \leftrightarrow ((|S| = 1) \wedge (\forall c \in MIN(\{c \in Lc | (\exists s \in S)(s < c)\}))(\exists n \in I^-)M(c, n)).$$

Theorems of the Algorithm of the Operation *Classify*

Theorem B.7. *Consider a task* $\langle Li, Lc, M, \langle I^+, I^- \rangle\rangle$ *with associated version space VS represented by* $\langle S, I^- \rangle$. *Then:*

$$(\forall i \in Li)((\forall c \in VS)M(c,i) \leftrightarrow (\forall s \in S)M(s,i)).$$

Proof. The proof coincides with that of theorem A.10. □

Theorem of the Algorithm of the Operation *Intersection*

Theorem B.8. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with version space* VS_1 *represented by* $\langle S_1, I_1^- \rangle$, *and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2 *represented by* $\langle S_2, I_2^- \rangle$. *Then the version space* $VS_{12} = VS_1 \cap VS_2$ *is represented by* $\langle S_{12}, I_{12}^- \rangle$ *where:*

$$S_{12} = \{c \in MinG(S_1, S_2) | (\forall n \in I_1^- \cup I_2^-) \neg M(c,n)\}$$
$$I_{12}^- = I_1^- \cup I_2^-$$

where $MinG(S_1, S_2) = MIN(\{c \in Lc | (\exists s_1 \in S_1)(\exists s_2 \in S_2)((s_1 \leq c) \wedge (s_2 \leq c))\})$.

Theorem of the Algorithm of the Operation *Subset?*

Theorem B.9. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with version space* VS_1 *represented by* $\langle S_1, I_1^- \rangle$, *and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2 *represented by* $\langle S_2, I_2^- \rangle$. *Then:*

$$(VS_1 \subseteq VS_2) \leftrightarrow ((\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1) \wedge (\forall n_2 \in I_2^-)(\forall c \in VS_1)\neg M(c, n_2)).$$

Proof. ($\rightarrow$) By theorem 5.21 $VS_1 \subseteq VS_2$ implies that:

$$(\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1). \tag{B.1}$$

By theorem 3.22 $VS_1 \subseteq VS_2$ implies that:

$$(\forall n_2 \in I_2^-)(\forall c \in VS_1)\neg M(c, n_2). \tag{B.2}$$

From formulas (B.1) and (B.2) the first part of the theorem is proven.

($\leftarrow$) Consider an arbitrary $c \in VS_1$. Since VS_1 is represented by $\langle S_1, I_1^- \rangle$, by theorem 5.21:

$$(\exists s_1 \in S_1)(s_1 \leq c).$$

Consider a particular $s_1 \in S_1$ such that $s_1 \leq c$. Since $(\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1)$ there exist $s_2 \in S_2$ such that $s_2 \leq s_1$. Thus,

$$s_2 \leq s_1 \leq c.$$

From transitivity of the relation "$\leq$" it follows that $s_2 \leq c$; i.e.,

$$(\exists s_2 \in S_2)(s_2 \leq c). \tag{B.3}$$

Since $(\forall n_2 \in I_2^-)(\forall c \in VS_1)\neg M(c, n_2)$:

$$(\forall n_2 \in I_2^-)\neg M(c, n_2). \tag{B.4}$$

By theorem 5.21 formulas (B.3) and (B.4) imply that:

$$c \in VS_2.$$

The second part of theorem is proven.

□

Theorem of the Algorithm of the Operation *Equal?*

Theorem B.10. *Consider a task* $\langle Li, Lc, M, \langle I_1^+, I_1^- \rangle\rangle$ *with version space* VS_1 *represented with* $\langle S_1, I_1^- \rangle$*, and a second task* $\langle Li, Lc, M, \langle I_2^+, I_2^- \rangle\rangle$ *with version space* VS_2 *represented with a pair* $\langle S_2, I_2^- \rangle$*. Then:*

$$(VS_1 = VS_2) \leftrightarrow$$
$$((S_1 = S_2) \wedge (\forall n_1 \in I_1^-)(\forall c \in VS_2)\neg M(c, n_1) \wedge (\forall n_2 \in I_2^-)(\forall c \in VS_1)\neg M(c, n_2)).$$

Proof. ($\rightarrow$) $VS_1 = VS_2$ implies $VS_1 \subseteq VS_2$ and $VS_1 \supseteq VS_2$. Thus, by theorem B.9:

$$(\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1) \tag{B.1}$$
$$(\forall s_2 \in S_2)(\exists s_1 \in S_1)(s_1 \leq s_2). \tag{B.2}$$
$$(\forall n_1 \in I_1^-)(\forall c \in VS_2)\neg M(c, n_1). \tag{B.3}$$
$$(\forall n_2 \in I_2^-)(\forall c \in VS_1)\neg M(c, n_2). \tag{B.4}$$

According to formula (B.1) consider an arbitrary $s_1 \in S_1$ and its corresponding $s_2 \in S_2$ such that $s_2 \leq s_1$. According to formula (B.2) for the same s_2 there exists $s_1' \in S_1$ such that $s_1' \leq s_2$. Thus, $s_1' \leq s_2 \leq s_1$. This implies $s_1' \leq s_1$. According to definitions 5.20 and 5.9 $s_1' \leq s_1$ holds when $s_1' = s_1$. Thus, $s_1 = s_2$. This means that

every $s_1 \in S_1$ is equal to exactly one element $s_2 \in S_2$. Analogously, by reversing the role of formulas (B.1) and (B.2) we can prove that: every $s_2 \in S_2$ is equal to exactly one element $s_1 \in S_1$. Consequently, $S_1 = S_2$. From the last derivation and formulas (B.3) and (B.4) the first part of the theorem is proven.

($\leftarrow$) From $S_1 = S_2$ we have that:

$$(\forall s_1 \in S_1)(\exists s_2 \in S_2)(s_2 \leq s_1) \tag{B.5}$$

$$(\forall s_2 \in S_2)(\exists s_1 \in S_1)(s_1 \leq s_2). \tag{B.6}$$

By theorem B.9 formula (B.5) and $(\forall n_2 \in I_2^-)(\forall c \in VS_1)\neg M(c, n_2)$ imply that $VS_1 \subseteq VS_2$. By the same theorem formula (B.6) and $(\forall n_1 \in I_1^-)(\forall c \in VS_2)\neg M(c, n_1)$ imply that $VS_2 \subseteq VS_1$. Thus, $VS_1 = VS_2$, and the second part of the theorem is proven.

□

Appendix C

Experimental Task

This appendix contains the concept-learning task used in the experiments with the instance-based boundary-set algorithms in chapter 6.

Given: (1) A 1-CNF instance language Li with 32 Boolean attributes.
(2) A 1-CNF instance language Li with 32 Boolean attributes.
(3) A 1-CNF concept language Lc with 32 Boolean attributes s.t $Li \subseteq Lc$.
(4) Set I^+ and I^- of positive and negative training instances:

+ (0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
+ (1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1)
− (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0)

Find: The IBBS of a version space specified by the sets I^+ and I^-.

References

Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, Vol. 6, No. 1, pp. 37–66. [165, 166]

Aha, D. (ed.) (1997). *Lazy learning*. Kluwer Academic Publishers, Norwell, MA. [165, 166]

Birkoff, G. (1973). *Lattice theory*, Vol. 25 of *AMS Colloquium Publications*. American Mathematical Society, Providence, third edition. First Edition, 1940. [71, 113]

Blake, C. and Merz, C. (1998). UCI Repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html. [4, 166, 174]

Cohen, P. and Feigenbaum, E. (eds.) (1981). *The handbook of artificial intelligence*, Vol. 3. Morgan Kaufmann, Los Altos, CA. [86, 147]

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, Vol. 13, pp. 21–27. [166]

Dasarathy, B. (ed.) (1991). *Nearest neighbor (NN) norm: NN pattern classification techniques*. IEEE Computer Society Press, Washington DC. [165, 166]

Domingos, P. and Pazzani, M. (1996). Beyond independence: conditions for the optimality of the simple bayesian classifier. *Proceedings of the Thirteenth International Conference on Machine Learning (ICML-96)*, pp. 105–112, Morgan Kaufmann, San Francisco, CA. [175]

Duda, R. and Hart, P. (1973). *Pattern classification and scene analysis*. John Wiley and Sons, New York, NY. [165, 166]

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, Vol. 13, No. 1, pp. 3–54. [184]

Goodrich, M. and Tamassia, R. (1998). *Data structures and algorithms in Java*. John Wiley and Sons, New York, NY. [20]

Gunter, C., Ngair, T., and Subramanian, D. (1997). The common order-theoretic structure of version spaces and ATMSs. *Artificial Intelligence*, Vol. 95, pp. 357–407. [73, 89]

Hart, P. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, Vol. 14, pp. 515–516. [165, 172]

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, Vol. 36, No. 2, pp. 177–221. [3, 4, 26, 86, 108, 149, 154]

Hirsh, H. (1989). *Incremental version-space merging: a general framework for concept learning*. Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA. [2, 4, 29, 79, 181, 193]

Hirsh, H. (1992a). The computational complexity of the candidate elimination algorithm. Technical report, Computer Science Department, Rutgers University, New Brunswick, NJ. [81]

Hirsh, H. (1992b). Polynomial-time learning with version spaces. *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 117–122, AAAI Press, Menlo Park, CA. [2, 3, 4, 20, 21, 26, 29, 32, 65, 66, 87, 89, 101, 103, 129, 154, 155, 199]

Hirsh, H. (1994). Generalizing version spaces. *Machine Learning*, Vol. 17, No. 1, pp. 5–45. [29]

Hirsh, H., Mishra, N., and Pitt, L. (1997). Version spaces without boundary sets. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 491–496, AAAI Press, Menlo Park, CA. [2, 4, 20, 21, 22, 29, 86, 108, 155]

Idemstam-Almquist, P. (1990). Demand networks: an alternative representation of version spaces. M.Sc. thesis, Department of Computer Science and Systems Sciences, The Royal Institute of Technology and Stockholm University, Stockholm, Sweden. [2, 4, 5, 20, 22, 32, 154, 188]

Kubat, M. (1991). Conceptual inductive learning: the case of unreliable teachers. *Artificial Intelligence*, Vol. 52, No. 2, pp. 169–182. [182]

Langley, P. (1996). *Elements of machine learning*. Morgan Kaufmann, San Francisco, CA. [1, 11, 15]

Mechelen, I. V., Hampton, J., Michalski, R., and Theuns, P. (1993). *Categories and concepts - theoretical views and inductive data analysis*. Academic Press, New York, NY. [1]

Mellish, C. (1991). The description identification problem. *Artificial Intelligence*, Vol. 52, No. 2, pp. 151–167.[182, 190]

Michalski, R. and Kodratoff, Y. (eds.) (1990). *Machine learning: an artificial intelligence approach*, Vol. 3. Morgan Kaufmann, Los Altos, CA.[1]

Michalski, R. and Tecuci, G. (eds.) (1994). *Machine learning : a multi-strategy approach*, Vol. 4. Morgan Kaufmann, San Francisco, CA.[1]

Michalski, R., Carbonell, J., and Mitchell, T. (eds.) (1983). *Machine learning: an artificial intelligence approach*, Vol. 1. Morgan Kaufmann, Los Altos, CA.[1]

Michalski, R., Carbonell, J., and Mitchell, T. (eds.) (1986). *Machine learning: an artificial intelligence approach*, Vol. 2. Morgan Kaufmann, Los Altos, CA.[1]

Mitchell, T. and Schwenger, G. (1978). A computer program for automated, empirical 13C NMR rule formation. *Organic Magnetic Resonance*, Vol. 11, No. 8, pp. 305–310.[3]

Mitchell, T. (1978). *Version spaces: an approach to concept learning*. Ph.D. thesis, Electrical Engineering Dept., Stanford University, Stanford, CA.[2, 3, 4, 9, 11, 14, 15, 16, 19, 20, 21, 26, 31, 32, 62, 65, 66, 67, 68, 72, 73, 74, 84, 103, 154, 180, 181, 193]

Mitchell, T. (1980). The need for biases in learning generalizations. Technical report, Computer Science Department, Rutgers University, New Brunswick, NJ.[16]

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, Vol. 18, No. 2, pp. 203–226.[2, 65]

Mitchell, T. (1997). *Machine learning*. McGraw-Hill, New York, NY.[1, 2, 3, 4, 14, 16, 19, 32, 74, 103, 166]

Murray, K. (1987). Multiple convergence: an approach to disjunctive concept acquisition. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 297–300, Morgan Kaufmann, Los Altos, CA. [3, 62, 182]

Plotkin, G. (1970). A note on inductive generalisation. *Machine Intelligence*, Vol. 5, pp. 153–163, Edinburgh University Press, Edinburgh, UK.[65]

Quinlan, J. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann, San Mateo, CA.[166, 175]

Russel, S. and Norvig, P. (1995). *Artificial intelligence: a modern approach*. Prentice Hall, Upper Saddle River, NJ.[16]

Sablon, G., DeRaedt, L., and Bruynooghe, M. (1994). Iterative versionspaces. *Artificial Intelligence*, Vol. 69, Nos. 1–2, pp. 393–410.[190]

Sablon, G. (1995). *Iterative versionspaces with an application in inductive logic programming*. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium.[4, 9, 54, 62, 182, 191]

Sebag, M. and Rouveirol, C. (1997). Tractable induction and classification in first order logic via stochastic matching. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 888–893, Morgan Kaufmann, San Francisco, CA.[4, 62]

Sebag, M. and Rouveirol, C. (2000). Resource-bounded relational reasoning: induction and deduction through stochastic matching. *Machine Learning*, Vol. 38, Nos. 1–2, pp. 41–62.[4, 62]

Sebag, M. (1996). Delaying the choice of bias: a disjunctive version space approach. *Proceedings of the Thirteenth International Conference on Machine Learning (ICML-96)*, pp. 444–452, Morgan Kaufmann.[4, 5, 62, 183]

Shavlik, J. and Dietterich, T. (eds.) (1990). *Readings in machine learning*. Morgan Kaufman, 1990, San Francisco, CA.[1]

Smirnov, E. and Braspenning, P. (1998a). Version space learning with instance-based boundary sets. *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98)*, pp. 460–464, John Wiley and Sons, Chichester, UK, (The Best Paper Prize).[105]

Smirnov, E. and Braspenning, P. (1998b). Version space retraction with instance-based boundary sets. *Proceedings of the Eight International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA-98)*, Vol. 1480 of *LNAI*, pp. 389–402, Springer, Berlin.[105, 120]

Smirnov, E. and Braspenning, P. (1999). Description identification and retraction with integrated instance/concept-based boundary sets. *Proceedings of the Tenth Midwest Artificial Intelligence and Cognitive Science Conference (MAICS-99)*, AAAI Technical Report CF-99-01, pp. 100–107, AAAI Press, Menlo Park, CA.[191]

Smirnov, E. and Herik, H. van den (2000). Applying preference biases to conjunctive and disjunctive version spaces. *Proceedings of the Ninth International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA-2000)*, LNAI 1904, pp. 321–330, Springer, Berlin, Germany.[184]

Smirnov, E. and Neves, J. (1998). A parametric generalised version space approach. *International Journal of Computer and Systems Sciences*, No. 5, pp. 708–728.[4, 62, 182]

Smirnov, E. (1992). Space fragmenting: a method for disjunctive concept acquisition. *Proceedings of the Fifth International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA-92)*, pp. 97–104, Elsevier, Dordrecht, The Netherlands. [62]

Smith, B. and Rosenbloom, P. (1990). Incremental non-backtracking focusing: a polynomially bounded generalization algorithm for version spaces. *Proceedings of the Eight National Conference on Artificial Intelligence (AAAI-90)*, pp. 848–853, MIT Press. [4, 155]

Subramanian, D. and Feigenbaum, J. (1986). Factorization in experiment generation. *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 518–522, Morgan Kaufmann, Los Altos, CA. [31]

Utgoff, P. (1984). *Shift of bias for inductive concept learning*. Ph.D. thesis, Computer Science Department, Rutgers University, New Brunswick, NJ. [16]

Vere, S. (1975). Induction of concepts in the predicate calculus. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 351–356. [65]

Wilson, D. and Martinez, T. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, Vol. 38, No. 3, pp. 257–286. [167, 181]

Wilson, D. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 2, pp. 408–421. [165, 173]

Winston, P. (1992). *Artificial Intelligence*. Addison-Wesley, Reading, MA, third edition. [155]

Summary

The thesis presents research on computational approaches to the concept-learning task. In this task, a learner receives training instances, each labelled as a positive or negative example of some unknown target concept. The goal is to find descriptions of the concept in a given concept language that accurately classify future unlabelled instances. To identify such descriptions the learner uses an acceptance criterion defined on the training data. One of the simplest acceptance criteria is the consistency criterion. It states that a description is consistent with the training instances of a target concept if the description correctly classifies those instances. When this criterion is employed the complete solution of the concept-learning task is a version space. Hence, the version space is defined as a set of all descriptions in the concept language that are consistent with the training instances of the target concept.

Version-space learning can be viewed as a search process in the concept language according to the constraints imposed by the training instances. The search is realised by the operations: *Initialise*, *Update*, and *Retraction*. The *Initialise* operation initialises the version space to be learned before the training instances are presented. Whenever a new training instance is presented the *Update* operation removes those descriptions from the version space that classify the instance incorrectly. Conversely, if a previously presented training instance is revoked the *Retraction* operation adds those descriptions to the version space that (1) classify the instance incorrectly, and (2) are consistent with the remaining training instances. The search process also requires the state-test operations *Converged?* and *Collapsed?*. The *Converged?* operation determines whether the version space consists of equivalent concept descriptions. The *Collapsed?* operation determines whether the version space is empty. If the version space is not empty, it can be used for instance classification. The instance classification is realised by the operation *Classify*. It is based on the rule of unanimous voting: an instance is classified if and only if all the descriptions in a version space agree on a classification of the instance. Since version spaces are sets, they are characterised by additional useful set operations *Member?*, *Intersection*, *Subset?* and *Equal?*.

The rule of unanimous voting can classify beyond training instances if the concept language is restricted and the target concept is represented in that language. The

concept language is restricted in the sense that it is incomplete. The concept language is incomplete if and only if at least one concept from the domain of discourse cannot be expressed in the language. Therefore, the inductive bias[1] of version spaces is a restriction bias.

In this thesis, we consider the definition of version spaces together with their basic operations (described above) as an abstract data type. Given a concept language, an implementation of the version-space abstract data type requires designing (1) a version-space representation, and (2) algorithms for the basic version-space operations based on that representation. A version-space representation is a finite data structure from which the version space to be learned can be derived. Given a concept language, for a version-space representation we distinguish two types of adequacy for the basic version-space operations. The representation is epistemologically adequate, if each operation can be implemented by an algorithm based on the representation. The representation is heuristically adequate, if the algorithm of each operation is tractable. So, the heuristical adequacy of a version-space representation is closely related to the tractability of the representation. A version-space representation is tractable for a concept language if it can be computed in time polynomial in the number of the training instances and the relevant properties of the language.

The standard version-space representation is by boundary sets. They are defined to contain the minimal and maximal descriptions of version spaces described in partially-ordered concept languages. Hence, the boundary sets delimit version spaces, although it is a well-known fact that they are intractable for most concept languages.

There are four problems in the domain of version spaces of which the intractability of the boundary sets is only one. We list the four problems below.

- **Problem with Incomplete Concept Languages:** if a target concept is not represented in a concept language, the version space to be learned is empty.
- **Computational Problem:** the standard version-space representation by boundary sets is intractable for most concept languages.
- **Retraction Problem:** most of the existing version-space representations are computationally inefficient for the operation *Retraction*.
- **Problem with Noisy Training Instances:** if at least one training instance is noisy, the description of the target concept is removed from the version space to be learned.

The problems described above are formulated in the problem statement investigated in this thesis:

[1] The inductive bias is a set of assumptions, that together with the training data, determine how a concept learner classifies future unlabelled instances.

Do there exist version spaces and their corresponding representations that can simultaneously handle:

(1) the problem with incomplete concept languages;

(2) the computational problem;

(3) the retraction problem; and

(4) the problem with noisy training instances?

In chapter 1 we formulate the problem statement. This thesis is an attempt to find answers to that problem statement. The next chapters describe them in detail.

In chapter 2 we formalise the concept-learning task. The task is viewed as a search task. In this context we introduce the notion of inductive bias.

In chapter 3 we propose to consider version spaces as an abstract data type. The data type is specified by the definition of version spaces and their set of basic operations. We analyse the properties of version spaces. Along with the main advantages two principal problems of version spaces are emphasised that correspond to the first and fourth part of the problem statement.

In chapter 4 we try to answer the first part of the problem statement. The key idea is to extend concept languages using either logical conjunction or disjunction. We prove that the conjunctive and disjunctive extensions of concept languages are more complete than the languages themselves (i.e., they can represent more concepts). This is the reason for introducing conjunctive and disjunctive version spaces. They are defined as version spaces in the conjunctive and disjunctive extensions of concept languages. We show that conjunctive and disjunctive version spaces are more complete than (ordinary) version spaces. Hence, they are solutions of more concept-learning tasks. In addition, we prove that conjunctive (disjunctive) version spaces are nonempty for all possible concept-learning tasks if and only if the conjunctive (disjunctive) extensions of concept languages are complete. Since the conjunctive and disjunctive extensions of concept languages are not necessarily complete, conjunctive and disjunctive version spaces are a partial solution to the problem of incomplete concept languages (the first part of the problem statement). Following the main line of chapter 3 we consider conjunctive and disjunctive version spaces as abstract data types. We prove that the conjunctive and disjunctive version-space abstract data types can be implemented using the version-space abstract data type.

In chapter 5 we consider the problem of implementing the version-space abstract data type. In this context we survey three existing version-space representations: (1) list representation, (2) boundary sets, and (3) unilateral boundary sets. They are presented in terms of their epistemological and heuristical adequacy for the basic version-space operations as well as in terms of their tractability. We improve the representations by developing algorithms of those basic version-space operations of which the implementation details were not considered previously. An analysis of the

representations and the algorithms of the operation *Retraction* leads to an adequate formulation of the problems that correspond to the second and third part of the problem statement.

In chapter 6 we answer the second and third part of the problem statement. The key idea is to consider the version space to be learned as an intersection of version spaces based on particular training instances. We prove that their boundary sets taken together delimit the version space. Therefore, the indexed families of these boundary sets are proposed as a version-space representation called instance-based boundary sets. We prove the epistemological adequacy of the instance-based boundary-set representation with respect to a subset of the basic version-space operations including the operation *Retraction* for the classes of intersection-preserving and union-preserving languages, introduced in the chapter. We derive the conditions for the tractability and the heuristical adequacy of the representation for the same subset of operations. A fair analysis of these conditions shows that they can easily be met in practice. Therefore, we conclude that the instance-based boundary-set representation simultaneously solves the computational problem and the retraction problem, that correspond to the second and third part of the problem statement. (This implies that the representation can be used for efficient implementations of the version-space abstract data type.)

In chapter 7 we answer the first, second, and third part of the problem statement. We show that the representations from chapters 5 and 6 can represent conjunctive and disjunctive version spaces and are epistemologically adequate with respect to the basic conjunctive and disjunctive version-space operations (for the concept languages for which they have been designed). Consequently, we conclude that the conjunctive and disjunctive version-space abstract data types can be implemented using each of these representations. Moreover, we determine when the representations are heuristically adequate for the basic conjunctive and disjunctive version-space operations, and when they tractably represent conjunctive and disjunctive version spaces. In this context, the conjunctive and disjunctive version spaces represented by instance-based boundary sets are considered as a solution to the problem with incomplete concept languages, the computational problem, and the retraction problem, that correspond to the first, second, and third part of the problem statement.

In chapter 8 we answer the fourth part of the problem statement. We propose to apply a particular classification algorithm, namely the k-nearest neighbour algorithm on the instance-based boundary sets of conjunctive and disjunctive version spaces. This results in two instance-based classification schemes. We show that their inductive bias is a preference bias since during the classification there exists a strong preference for certain concept descriptions over others. Nevertheless, the schemes do not require any change in the instance-based boundary-set representation of conjunctive and disjunctive version spaces. Hence, they can be added to the conjunctive and disjunctive version-space abstract data types realised with instance-based boundary sets. We test the systems, based on the classification schemes, on

standard datasets. We note that their classification performance is comparable with that of the C4.5 decision-tree learner. In addition, we combine the systems with the condensed nearest-neighbour algorithm and the edited nearest-neighbour algorithm. The first combination aims at reduction of the computational complexity, but the experimental results show a significant drop in the classification performance. The second combination aims to achieve robustness of the classification performance with respect to noise in training instances. Such a robustness is confirmed by experimental results.

Next to the instance-based classification schemes, described above, in chapter 8 we consider a second approach for applying preference biases with instance-based boundary sets although it is only briefly sketched. The approach is a methodology for designing a broad spectrum of separate-and-conquer algorithms based on instance-based boundary sets.

From these results we conclude that the instance-based boundary sets can be used for designing learning procedures that implement different types of inductive biases including those that can handle noise in training instances. Thus, the fourth part of the problem statement is solved.

Chapter 9 ends the thesis. A proper evaluation of the problem statement is presented. In summary, it reads:

If the concept languages are intersection-preserving or union-preserving, the conjunctive and disjunctive version spaces represented by instance-based boundary sets can solve simultaneously:

(1) the problem with incomplete concept languages;

(2) the computational problem;

(3) the retraction problem; and

(4) the problem with noisy training instances.

Samenvatting

Dit proefschrift beschrijft onderzoek naar computationele benaderingen om een bepaald begrip (een concept) te leren. Een lerende agent krijgt een aantal oefenvoorbeelden voorgelegd die elk zijn voorzien van een label dat aangeeft of een voorbeeld wel of niet tot het concept behoort. De agent zoekt op basis van de verzameling oefenvoorbeelden naar een goede beschrijving in een gegeven concepttaal. Het doel is om beschrijvingen te vinden die de agent in staat stelt om toekomstige voorbeelden (die niet zijn voorzien van een label) correct te classificeren.

Hiertoe zoekt de agent naar beschrijvingen in de concepttaal die voldoen aan een acceptatiecriterium. Een eenvoudig voorbeeld van een dergelijk criterium is het consistentiecriterium. Volgens dit criterium is een beschrijving consistent met de voorbeelden behorende bij een te leren concept indien de beschrijving de oefenvoorbeelden correct classificeert. Een op basis van het consistentiecriterium verkregen oplossing van de taak om een concept te leren vormt een zogeheten versieruimte. Een versieruimte is de verzameling van alle beschrijvingen die consistent zijn met de oefenvoorbeelden van het te leren concept.

Het leren van concepten via versieruimten kan worden beschouwd als het zoeken naar een beschrijving die in overeenstemming is met de oefenvoorbeelden. Het zoekproces wordt uitgevoerd in termen van de operaties: *Initialise*, *Update*, en *Retraction*. De operatie *Initialise* initialiseert de versieruimte alvorens de oefenvoorbeelden worden aangeboden. Zodra er een gelabeld voorbeeld wordt aangeboden, verwijdert de operatie *Update* alle beschrijvingen in de versieruimte die inconsistent zijn met het oefenvoorbeeld. Wanneer een eerder aangeboden oefenvoorbeeld wordt ingetrokken dan voegt de operatie *Retraction* alle beschrijvingen toe die dat voorbeeld als incorrect classificeren en die tevens consistent zijn met de overige oefenvoorbeelden.

Daarnaast maakt het zoekproces gebruik van twee testoperaties: *Converged?* en *Collapsed?*. De operatie *Converged?* bepaalt of de versieruimte bestaat uit equivalente conceptbeschrijvingen. De operatie *Collapsed?* bepaalt of de versieruimte leeg is. Indien de versieruimte niet leeg is kan deze gebruikt worden voor de classificatie van voorbeelden middels de operatie *Classify*. Deze operatie hanteert de regel van eenstemmigheid. Volgens deze regel wordt (i) een voorbeeld als positief geclassificeerd (zoals behorend tot het geleerde concept) wanneer alle beschrijvingen in de

versieruimte consistent zijn met het voorbeeld; (ii) een voorbeeld als negatief geclassificeerd (d.w.z., niet behorend tot het geleerde concept) wanneer alle beschrijvingen in de versieruimte inconsistent zijn met het voorbeeld; en (iii) een voorbeeld als niet classificeerbaar beschouwd in alle andere gevallen. Aangezien versieruimten verzamelingen zijn, gebruiken we nog een aantal bruikbare operaties op verzamelingen, namelijk *Member?*, *Intersection*, *Subset?* en *Equal?*.

De regel van eenstemmigheid kan voorbeelden buiten de oefenvoorbeelden classificeren als de concepttaal beperkt is en het beoogde concept gerepresenteerd wordt in die taal. De concepttaal is beperkt in de zin dat deze onvolledig is. Een concepttaal is onvolledig dan en slechts dan als minstens één concept uit het domein niet in die taal kan worden uitgedrukt. Daarom is de inductieve bias[2] van een versieruimte een restrictie-bias.

In dit proefschrift worden de definitie van versieruimten alsmede de bijbehorende operaties beschouwd als een abstract datatype. Bij een gegeven concepttaal vereist de implementatie van het abstracte datatype van een versieruimte de specificatie van (1) een versieruimterepresentatie, en (2) algoritmen voor de elementaire versieruimte-operaties. Een versieruimterepresentatie is een eindige datastructuur in de concepttaal waaruit de te leren versieruimte kan worden afgeleid. Bij een gegeven concepttaal gebruiken we twee manieren om adequaatheid van de versieruimte-operatoren aan te geven: epistemologisch adequaat en heuristisch adequaat. Een representatie is epistemologisch adequaat indien iedere operatie geïmplementeerd kan worden door een algoritme dat gebaseerd is op die representatie. Een representatie is heuristisch adequaat indien iedere operatie geïmplementeerd kan worden door een algoritme dat "tractable" is.

Het heuristisch adequaat zijn van een versieruimterepresentatie voor de elementaire versieruimte-operaties hangt nauw samen met de "tractability" van de representatie. Een versieruimterepresentatie is "tractable" voor een concepttaal indien deze kan worden berekend voor alle mogelijke verzamelingen van de geleerde voorbeelden in een tijd die polynomiaal is met betrekking tot de omvang van de oefenvoorbeelden en de relevante eigenschappen van de concepttaal.

"Boundary sets" vormen de standaardrepresentatie voor versieruimten. Bij deze representatie wordt uitgegaan van partieel geordende concepttalen. "Boundary sets" bevatten de minimale en maximale beschrijvingen van versieruimten. "Boundary sets" bakenen versieruimten af, hoewel het algemeen bekend is dat zij voor de meeste concepttalen "intractable" zijn. Het domein van de versieruimten kent vier problemen waarvan de "intractability" van de "boundary-set" representaties er één is. De vier problemen zijn:

- **Het probleem van onvolledige concepttalen:** als een te leren concept niet in een concepttaal gerepresenteerd is, dan is de bijbehorende versieruimte leeg.

[2] De inductieve bias is een verzameling aannamen die, tezamen met de geleerde voorbeelden, bepalen op welke wijze ongelabelde (niet geleerde) voorbeelden worden geclassificeerd.

- **Het computationele probleem:** de standaard-versieruimterepresentatie in termen van "boundary sets" is niet "tractable" voor de meeste concepttalen.
- **Het retractieprobleem:** de meeste bestaande versieruimterepresentaties zijn computationeel inefficiënt met betrekking tot de operatie *Retraction*.
- **Het probleem van ruis in de oefenvoorbeelden:** als de te leren voorbeelden ruis bevatten, worden beschrijvingen van het te leren concept verwijderd uit de versieruimte.

Deze vier problemen zijn vervat in de centrale probleemstelling van dit proefschrift:

Bestaan er versieruimten en bijbehorende representaties die tegelijkertijd een oplossing bieden aan de volgende vier problemen:

(1) Het probleem van onvolledige concepttalen;

(2) Het computationele probleem;

(3) Het retractieprobleem; en

(4) Het probleem van ruis in de oefenvoorbeelden?

Dit proefschrift poogt een antwoord te vinden op deze probleemstelling. De structuur van het proefschrift is als volgt. Hoofdstuk 1 geeft een introductie en beschrijft de probleemstelling.

Hoofdstuk 2 formaliseert de taak om een begrip (concept) te leren. De taak wordt beschouwd als een zoektaak. Het begrip inductieve bias wordt geïntroduceerd.

In hoofdstuk 3 worden versieruimten beschreven in termen van een abstract datatype. Het datatype wordt gespecificeerd in termen van de definitie van versieruimten en de bijbehorende verzameling van elementaire operaties. De eigenschappen van versieruimten worden geanalyseerd. Naast een bespreking van de voornaamste voordelen, worden twee problemen van versieruimten nader uitgewerkt: het probleem van onvolledige concepttalen (deel 1 van de probleemstelling) en het probleem van ruis in de voorbeelden (deel 4 van de probleemstelling).

Hoofdstuk 4 richt zich op het probleem van de onvolledige concepttalen. Het centrale idee is om de concepttaal uit te breiden met behulp van logische conjuncties of disjuncties. Er wordt bewezen dat conjunctieve en disjunctieve uitbreidingen van een concepttaal volledige zijn dan de oorspronkelijke concepttaal (d.w.z., zij bevatten meer concepten). Derhalve worden conjunctieve en disjunctieve versieruimten geïntroduceerd. Deze versieruimten zijn gedefinieerd als versieruimten die gebaseerd zijn op de conjunctieve en disjunctieve uitbreidingen van de concepttaal. Vervolgens wordt bewezen dat conjunctieve en disjunctieve versieruimten volledige zijn dan (gewone) versieruimten. De uitgebreide versieruimten zijn daarom oplossingen

voor een groter aantal concept-leertaken. Bovendien wordt bewezen dat conjunctieve (disjunctieve) versieruimten niet leeg zijn voor alle mogelijke concept-leertaken dan en slechts dan als de conjunctieve (disjunctieve) uitbreidingen van de concepttalen volledig zijn. Aangezien de conjunctieve en disjunctieve uitbreidingen van concepttalen niet noodzakelijkerwijs volledig zijn, vormen conjunctieve en disjunctieve versieruimten enkel een gedeeltelijke oplossing voor het probleem van de onvolledige concepttalen. In aansluiting op hoofdstuk 3 worden conjunctieve en disjunctieve versieruimten beschouwd als abstracte datatypen. Er wordt bewezen dat de conjunctieve en disjunctieve versieruimten geïmplementeerd kunnen worden met behulp van het abstracte datatype voor versieruimten.

Hoofdstuk 5 richt zich op het probleem van de implementatie van het abstracte datatype voor versieruimten. Er worden drie bestaande versieruimterepresentaties onderzocht: (1) lijstrepresentaties, (2) "boundary sets", en (3) "unilaterale boundary sets". De representaties worden gepresenteerd in termen van de begrippen epistemologisch adequaat en heuristisch adequaat voor de ("tractability" van) elementaire versieruimte-operaties. Deze representaties worden verbeterd door het ontwikkelen van algoritmen voor elementaire versieruimte-operaties waarvan de implementatie nog niet eerder werd onderzocht. Analyse van de representaties en bijbehorende algoritmen van de operatie *Retraction* leiden tot een adequate formulering van het computationele probleem (deel 2 van de probleemstelling) en het retractieprobleem (deel 3 van de probleemstelling).

Hoofdstuk 6 behandelt het computationele probleem en het retractieprobleem. Het stelt een nieuwe versieruimterepresentatie voor die kan worden gebruikt voor de implementatie van het abstracte datatype van versieruimten. Het centrale idee is om de te leren versieruimte te beschouwen als een doorsnijding van versieruimten die gebaseerd zijn op specifieke verzamelingen oefenvoorbeelden. Er wordt bewezen dat de gezamenlijke "boundary sets" de versieruimte afbakenen. Vervolgens wordt de "instance-based boundary-set" representatie voorgesteld. Deze versieruimterepresentatie bestaat uit geïndexeerde families van "boundary sets". Met betrekking tot een deelverzameling van de elementaire versieruimte-operaties wordt bewezen dat de "instance-based boundary-set" representatie epistemologisch adequaat is. De deelverzameling bevat de operatie *Retraction* voor de klassen van "intersection-preserving" en "union-preserving" talen. Daarnaast worden de voorwaarden voor de "tractability" en het heuristisch adequaat zijn van de representaties voor dezelfde deelverzameling afgeleid. Uit een analyse van deze voorwaarden blijkt dat er in praktische situaties gemakkelijk aan voldaan kan worden. Om die reden worden "instance-based boundary sets" beschouwd als een oplossing voor zowel het computationele probleem als het retractieprobleem.

Hoofdstuk 7 beantwoordt de eerste drie delen van de vraagstelling. Er wordt aangetoond dat de in de hoofdstukken 5 en 6 voorgestelde representaties geschikt zijn voor conjunctieve en disjunctieve versieruimten en dat zij tevens epistemologisch adequaat zijn met betrekking tot de elementaire conjunctieve en disjunctieve

versieruimte-operaties (voor de concepttalen waarvoor zij zijn ontworpen). Hieruit wordt geconcludeerd dat de abstracte datatypen van conjunctieve en disjunctieve versieruimten kunnen worden geïmplementeerd met behulp van deze representaties. Bovendien wordt bepaald wanneer de representaties heuristisch adequaat zijn voor de elementaire conjunctieve en disjunctieve versieruimte-operaties, en wanneer zij "tractable" conjunctieve en disjunctieve versieruimten representeren. Conjunctieve en disjunctieve versieruimten, geïmplementeerd middels "instance-based boundary sets", worden beschouwd als een oplossing voor het probleem van onvolledige concepttalen, het computationele probleem, en het retractieprobleem.

Hoofdstuk 8 behandelt het probleem van ruis in de oefenvoorbeelden. Hiertoe wordt voorgesteld om het "k-nearest neighbour" classificatie-algoritme toe te passen op de "instance-based boundary sets" van conjunctieve en disjunctieve versieruimten. Dit leidt tot twee "instance-based" classificatieschema's. Er wordt aangetoond dat de inductieve bias van deze schema's een preferentie-bias is. Gedurende classificatie wordt de voorkeur gegeven aan bepaalde conceptbeschrijvingen boven andere. Niettemin vereisen geen van beide schema's aanpassingen aan de "instance-based boundary-set" representaties van de conjunctieve en disjunctieve versieruimten. Om die reden kunnen zij worden toegevoegd aan de abstracte datatypen van de conjunctieve en disjunctieve versieruimten die zijn geïmplementeerd met "instance-based boundary sets".

Classificatiesystemen, gebaseerd op beide schema's, zijn getest op standaarddatasets. Uit de resultaten blijkt dat de prestatie vergelijkbaar is met de prestatie van het op een beslisboom gebaseerde C4.5-classificatiesysteem. Daarnaast zijn de systemen gecombineerd met het "condensed nearest neighbour" algoritme en het "edited nearest neighbour" algoritme. De eerste combinatie beoogt een reductie van de computationele complexiteit, maar levert een aanzienlijk lagere prestatie op. De tweede combinatie beoogt een meer robuuste classificatie in geval van ruis in de voorbeelden. De experimentele resultaten bevestigen dat het gecombineerde systeem beter bestand is tegen ruis in de oefenvoorbeelden.

Naast de hierboven beschreven "instance-based" classificatieschema's, wordt in het kort een tweede aanpak geschetst voor het toepassen van preferentie-biases bij "instance-based boundary sets". De aanpak is een methodologie voor het ontwerpen van een breed spectrum van verdeel-en-heers-algoritmen die zijn gebaseerd op "instance-based boundary sets".

Dit alles leidt tot de conclusie dat "instance-based boundary sets" kunnen worden gebruikt voor het ontwerpen van leerprocedures met verschillende typen van inductieve biases, waaronder die voor het omgaan met ruis in de oefenvoorbeelden.

Hoofdstuk 9 besluit met het beantwoorden van de probleemstelling:

Indien de onderliggende concepttalen "intersection-preserving" of "union-preserving" zijn, lossen conjunctieve en disjunctieve versieruimten gerepresenteerd door "instance-based boundary sets" de volgende vier problemen op:

(1) Het probleem van onvolledige concepttalen;

(2) Het computationele probleem;

(3) Het retractieprobleem; en

(4) Het probleem van ruis in de oefenvoorbeelden.

Curriculum Vitae

Evgueni Nikolaevich Smirnov was born in Sofia, Bulgaria, on the eleventh of August 1965. From 1979 he attended the Mathematical Gymnasium in Sofia. After graduation in 1983, he began his study of Computer Science at the Technical University of Sofia, specialising in Artificial Intelligence. In 1988 he successfully defended his M.Sc. thesis in the field of expert systems. Between 1988 and 1996 he worked in the Artificial Intelligence laboratories of the Technical University of Sofia, the Institute of Informatics (Sofia), the Institute of Information Technologies (Sofia) (Bulgarian Academy of Science), and the Universiteit Maastricht (Maastricht, The Netherlands). His research interest was in the fields of expert systems, concept learning, version spaces, and multiple explanation-based learning. In 1996 he successfully defended his doctoral thesis. After the defence he accepted a seven-month postdoctoral position at the Artificial Intelligence laboratory of the University of Minho (Braga, Portugal).

In 1997 he was invited as an assistant professor at the Department of Computer Science, Universiteit Maastricht, The Netherlands. Besides his teaching activities he continued the research on version spaces. The new results he obtained were published in a paper that received the Best Paper Prize of the Thirteenth European Conference on Artificial Intelligence (ECAI-98), Brighton, UK. This was a natural start of a new Ph.D. research that resulted in several publications and this thesis. Currently, he has a postdoctoral position from the Netherlands Organisation for Scientific Research (NWO).

SIKS Dissertatiereeks

In 2001 zijn de volgende SIKS-disertaties verschenen.

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
Promotores: Prof. dr. J.-J.Ch. Meyer (UU)
Prof. dr.ir. L.C. van der Gaag (UU)
Co-promotor: Dr. C.L.M. Witteman (UU)
Promotie: 12 maart 2001

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
Promotor: Prof. dr. J.-J.Ch. Meyer (UU)
Co-Promotoren: Dr. W. van der Hoek (UU)
Dr. F.S. de Boer (UU)
Promotie: 5 februari 2001

2001-3 Maarten van Someren (UvA)
Learning as problem solving
Promotor: Prof. dr. B.J. Wielinga (UvA)
Promotie: 1 maart 2001

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
Promotor: Prof. dr. H.J. van den Herik (UM)
Promotie: 22 februari 2001